AFRL-IF-RS-TR-2005-50
Final Technical Report
February 2005

CAMERA: COORDINATION AND MANAGEMENT ENVIRONMENTS FOR RESPONSIVE AGENTS

University of Southern California at Marina Del Rey

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. J133, H361

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

STINFO FINAL REPORT

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2005-50 has been reviewed and is approved for publication

APPROVED: /s/

DANIEL E. DASKIEWICH
Project Engineer

FOR THE DIRECTOR: /s/

JAMES A. COLLINS, Acting Chief
Advanced Computing Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	FEBRUARY 2005	Final Jun 99 – Nov 03

4. TITLE AND SUBTITLE
CAMERA: COORDINATION AND MANAGEMENT ENVIRONMENTS FOR RESPONSIVE AGENTS

5. FUNDING NUMBERS
C - F30602-99-1-0524
PE - 62301E
PR - H361
TA - 01
WU - 01

6. AUTHOR(S)
Robert Neches and
Pedro Szekely

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
University of Southern California at Marina Del Rey
Information Sciences Institute
4676 Admiralty Way
Marina Del Rey California 90292-6695

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington Virginia 22203-1714

AFRL/IFT
525 Brooks Road
Rome New York 13441-4505

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2005-50

11. SUPPLEMENTARY NOTES
AFRL Project Engineer: Daniel E. Daskiewich/IFT/(315) 330-7731/ Daniel.Daskiewich@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 Words)***
The CAMERA project produced the SNAP (Schedules Negotiated by Agent-based Planners) flight scheduling software in experimental use by Harrier squadrons of Marine Air Group 13, stationed in Yuma, AZ; it was also fielded aboard the USS Bonhomme Richard, the USS Belleau Wood, the USS Pelleliu and the USS Essex that conducted operations in Iraq, Japan and Afghanistan. The CAMERA project also produced an open hybrid solver architecture that allows off-the-shelf solvers to be combined to solve a problem. Finally, it produced a family of market-inspired negotiation algorithms called MARBLES.

14. SUBJECT TERMS
Automated Negotiation, Multi-Agent Systems, Agent Management Environments, Human-In-The-Loop Flight Scheduling

15. NUMBER OF PAGES
52

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

List of Figures

1 Executive Summary

The CAMERA project (Coordination and Management Environments for Responsive Agents) investigated negotiation-based approaches to large and difficult scheduling problems, in the context of real-world challenges offered by Harrier aircraft operations for the United States Marine Corps (USMC).

This work led to practical demonstrations which had significant impact upon USMC, as evidenced by the following quotes, and absorption into the CACE ACTD (Coherent Analytical Computing Environment Advanced Concept Technology Demonstration), which in turn received a positive evaluation from its users which led to USMC establishment of a program of record, also called CACE, for deployment of the technology:

> *"…gives a first-time opportunity to do extremely complex balancing of considerations within critical time constraints…will provide 'look-ahead' capabilities never before available. I think you have uncorked a genie."*
>
> *- M.A. Hough, Major General, USMC*

> *"relevant to any set of specialized data systems…can enhance their singular value through cross functional negotiation based on guidance in support of a common intent."*
> *- Colonel D.L. Buland, Commanding Officer, Marine Air Group 13*

Details of the test problem and results are provided in Section 2.

The effort yielded significant research results, as well, which are extensively reviewed in Section 3.

2 The Logistics Challenge Problem

The CAMERA project focused on developing negotiation technology to solve real, practical problems. The project focused on a logistics challenge problem, which involved the coordinated scheduling of flight operations and aircraft maintenance. The Marine Air Group 13 (MAG-13) in Yuma, Arizona provided the context for this challenge problem. They provided the requirements, the data and military personnel committed to evaluate and use incremental releases of the software. The logistics challenge problem is a practical problem that the Marines solve on a daily basis to schedule their flight operations and maintenance.

2.1 Context

2.1.1 The Users

The users involved in producing the flight and maintenance schedules fall into three groups:

Commanding officers: The commanding officers define the overall goals of the schedule for a given planning horizon. For example, participate in a specific training exercise, have Smith and Jones obtain their night systems qualification, fly 20 sorties per day and maintain flight equity (every pilot flies roughly the same number of hours per month).

Operations officers and staff: The operations people are in charge of figuring out how to carry out the commander's intent. They must figure out who flies what type of mission, when, and where.

Maintenance officers and staff: The maintenance people must figure out what type of maintenance to do on each aircraft so that they can support the flights that the operations people want to fly. They must also deal with unscheduled maintenance requirements that arise when aircraft break down.

The commanding officers generate yearly and monthly guidance. The operations people must produce weekly schedules to meet the guidance, and refine those schedules every day to produce the schedules that get executed every day. The operations and maintenance people must communicate frequently to coordinate their schedules: in order for operations to produce a schedule they need to know how many aircraft of each type are available. In order to answer that question maintenance needs to know the flight schedule in order to figure out whether they have time to carry out all the usage-based and calendar-based maintenance that must be done on the aircraft. Usage-based maintenance is performed after the aircraft has accumulated a number of flight hours; calendar-based maintenance is performed after a predefined number of days pass – even if the aircraft had not been flown at all. The cycle is broken by starting with estimates and refining those estimates though iterative refinement of the schedules.

2.1.2 The Current Way of Doing It

The users have databases that record all the information relevant for producing the schedules. Operations has databases that record the flight logs of each pilot, their qualifications, etc. Maintenance has databases that record all the maintenance work items that must be performed or are being performed on each aircraft.

These databases have viewers and editors that allow users to see what is going on, and to edit the information. Operations has a sophisticated schedule editor application that enables users to enter and validate flight schedules, but they do not have a scheduling application that computes a schedule. The users determine the schedules manually, typically on a white board, and then enter it in the computer to validate it and to print the official schedules that get signed by the commanding officers. The maintenance

schedules are kept on white board and paper and never entered in a computer.

2.1.3 The Logistics Challenge Goal

The goal of the logistics challenge is to automate the production of the operations and maintenance schedules. The payoff is large because developing the schedules by hand is labor intensive and time consuming (typically 6 hours for a weekly operations schedule), and little time is left to explore alternatives and to deal with often changing requirements. In addition, producing schedules that extend beyond a week is infeasible, resulting in commanders having limited ability to forecast the consequences of taking on new commitments (e.g., can you participate in a week-long exercise at the beginning of next month and still make your deployment commitments 6 months from now?).

The University of Southern California Information Sciences Institute's CAMERA project focused on the operations side of the logistics challenge problem. Vanderbilt University's sister project - also funded by the Defense Advanced Research Projects Agency's (DARPA's) Autonomous Negotiation Teams (ANTs) program - focused on the maintenance side of the problem. The two institutions collaborated to build a coordinated solution to the integrated operations/maintenance logistics challenge problem.

This report focuses on the operations aspect of the challenge problem.

2.2 The Operations Problem

We describe the operations side of the logistics challenge in terms of the goals that commanders can specify, in terms of the objective functions that define the score of a schedule, the resources and tasks, and constraints that define the scheduling problem. Screen-shots of the Schedules Negotiated by Agent-based Planners (SNAP) flight scheduling application are included to show how the different aspects of the problem are handled in the application. The next sections will discuss the underlying CAMERA negotiation technology.

2.2.1 Goals

The following list of goals is not a complete list of all the goals that commanders would like to state. The list contains the goals that the challenge problem focused on.

Acquire qualification: A goal for a pilot to acquire a specific skill such as night systems, or carrier operation. Each skill requires the pilot to fly a specific sequence of mission codes with an instructor pilot who is qualified to evaluate performance, e.g., Jones gain night systems qualification.

Figure 1 shows the SNAP screen where operators can define the qualification goals for pilots.

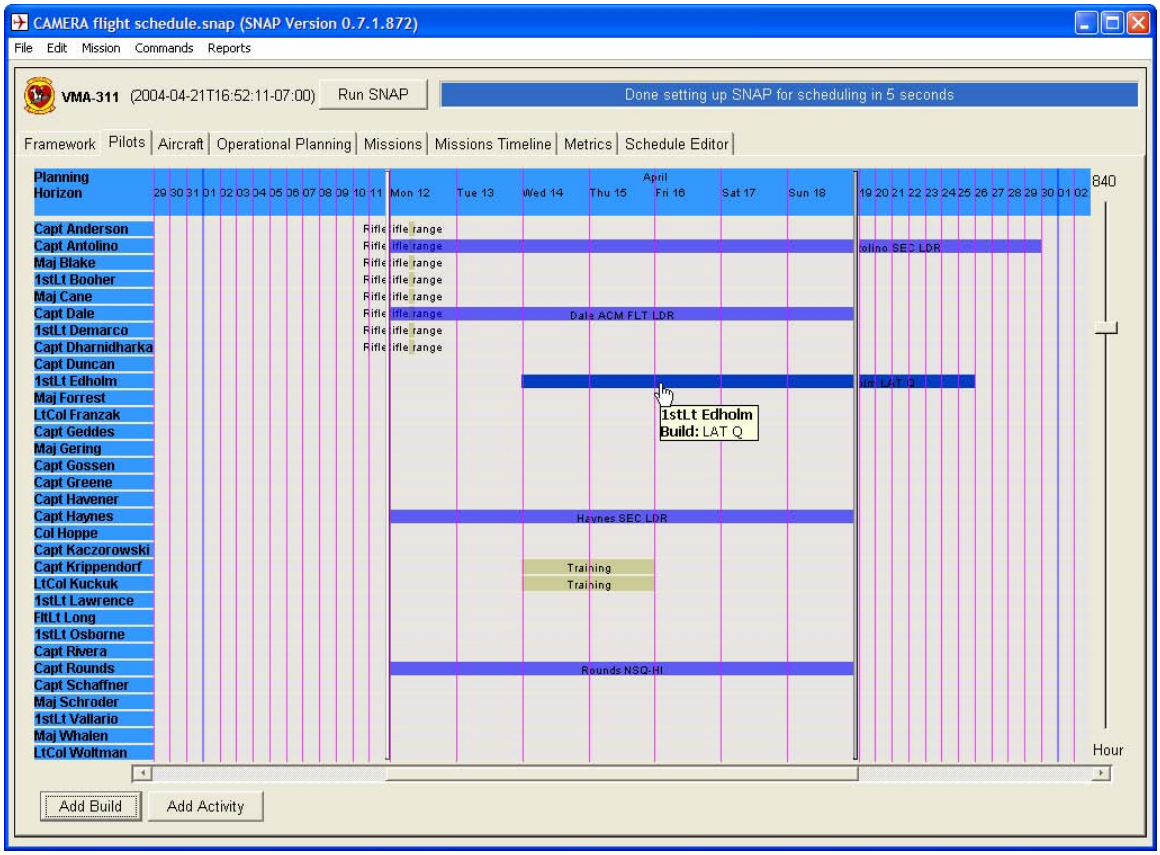


Figure 1: Defining the qualification goals for pilots

Achieve and maintain core competency: Pilots achieve core competencies such as air-to-air or air-to-ground by flying a specific sequence of mission codes. Because competency degrades with time, pilots must periodically fly specific subsets of the core competency codes to maintain their competency. Commanders can specify goals for certain pilots to achieve certain core competencies during a period.

Figure 2 shows the screens for defining core competency goals (also referred to as squadron foci). The screen shows the level to which different pilots have achieved competency in a particular skill (Low Altitude Tactics). SNAP shows whether pilots are current on a training code (green cells), whether their skill on a training code has expired (red) and whether they don't have a particular skill, but are eligible to obtain it (white cells). The operator can request that a certain number of sorties on the schedule be devoted to improve a skill, and the system will automatically schedule sorties to refresh or obtain skills as required.

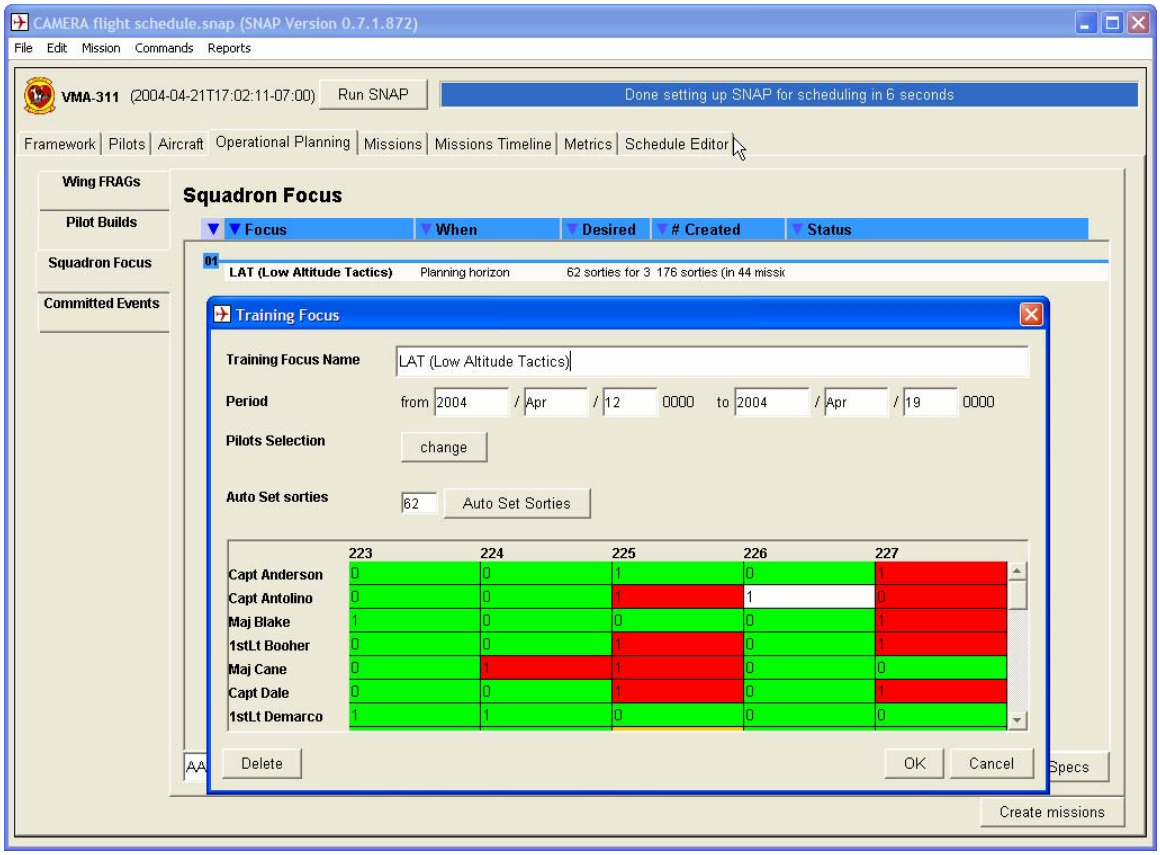


Figure 2: Defining a squadron focus and allocating focus sorties to individual pilots

Fulfill specific sorties: This goal specifies that the squadron must send a specified number of aircraft with pilots capable of flying specific missions to specified locations and at specified times. War-time sorties are examples of this goal. E.g., 2 close air support aircraft at 12:30 am at 29 Palms.

Figure 3 shows a screen shot of the Operational Planning screen where the operator can define these sorties. The screen shows that the user has defined 4 such sorties, and is editing the first one.

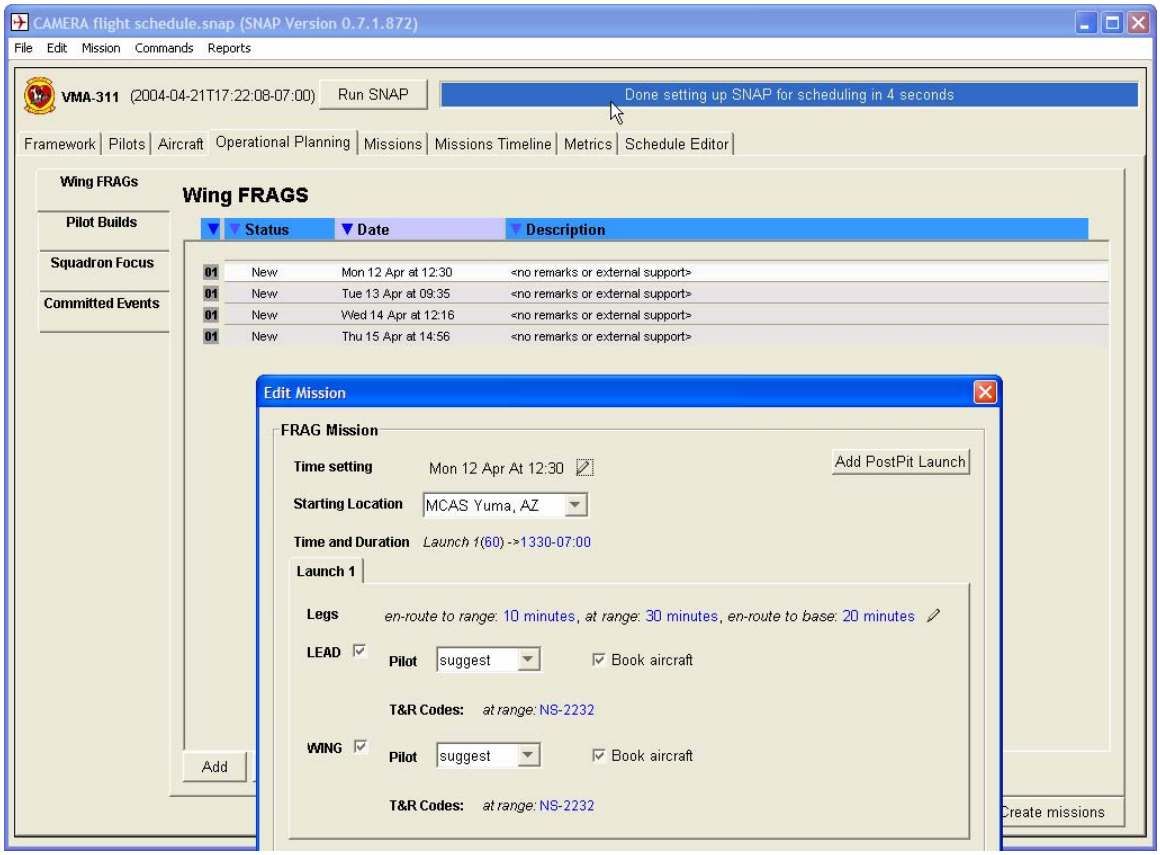


Figure 3: Entering "FRAGs", Wing-ordered training or war-time missions

Schedule a given number of sorties: Commanders can specify that a certain number of sorties be flown during a period, e.g., fly 100 sorties next week.

Schedule a given number of sorties per pilot per period: Specifies the desired number of sorties that specific pilots should fly during specific periods, e.g., Smith to fly 5 sorties per week. Typically flying less is bad, and flying more is wasteful.

Minimally modify an existing schedule to repair a schedule with respect to changes in resources or requirements: Specifies that a new schedule should be as similar as possible to a given schedule.

Figure 4 shows a screen shot of the display where operators can specify the last three types of goals (in addition to variations of these goals). Each row corresponds to a separate goal (also referred to as an objective). The table at the right of the image shows a history of how well different versions of the schedules satisfy the goals. In this example the operator is getting close to a satisfactory schedule. The schedule contains 2 more sorties than desired, and one pilot is flying too much.

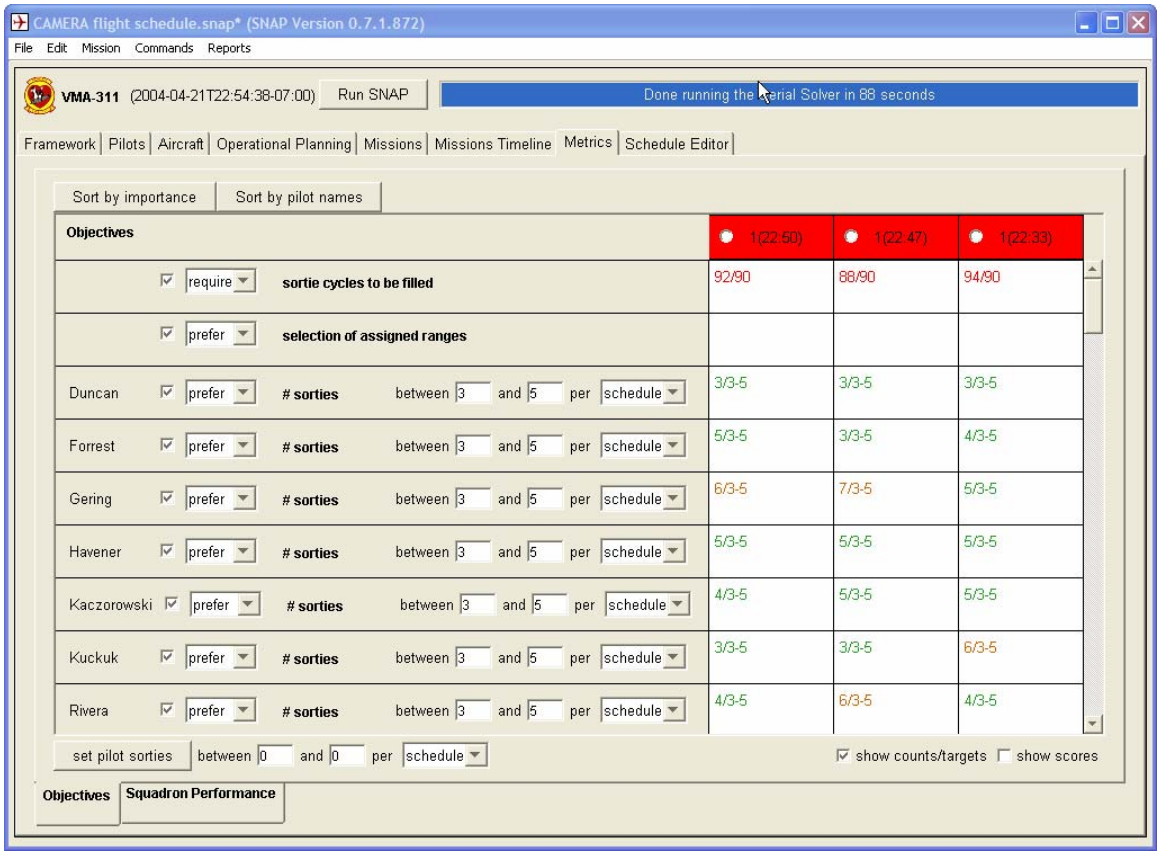


Figure 4: Fine-tuning scheduling preferences in the Metrics screen

2.2.2 Objective Function

The objective function is a linear combination of metrics that specify how well each goal is achieved. Each goal is scored using a piece-wise linear function that specifies how good it is to obtain a certain quantity of something. For example, an objective function for the number of sorties goal could be specified as follows: flying 0 sorties gives a score of 0, flying 90% of the sorties gives a score of 0.5, and flying 100% gives a score of 1.

The score of a schedule is specified as a linear combination of the score for each goal. Note: in the implemented system, the user interface did not allow users to give numeric weights. Instead we offered the values "require", "prefer" and "don't care", which were defined in such a way that all the "prefers" weighed less than a single "require" (see Figure 4).

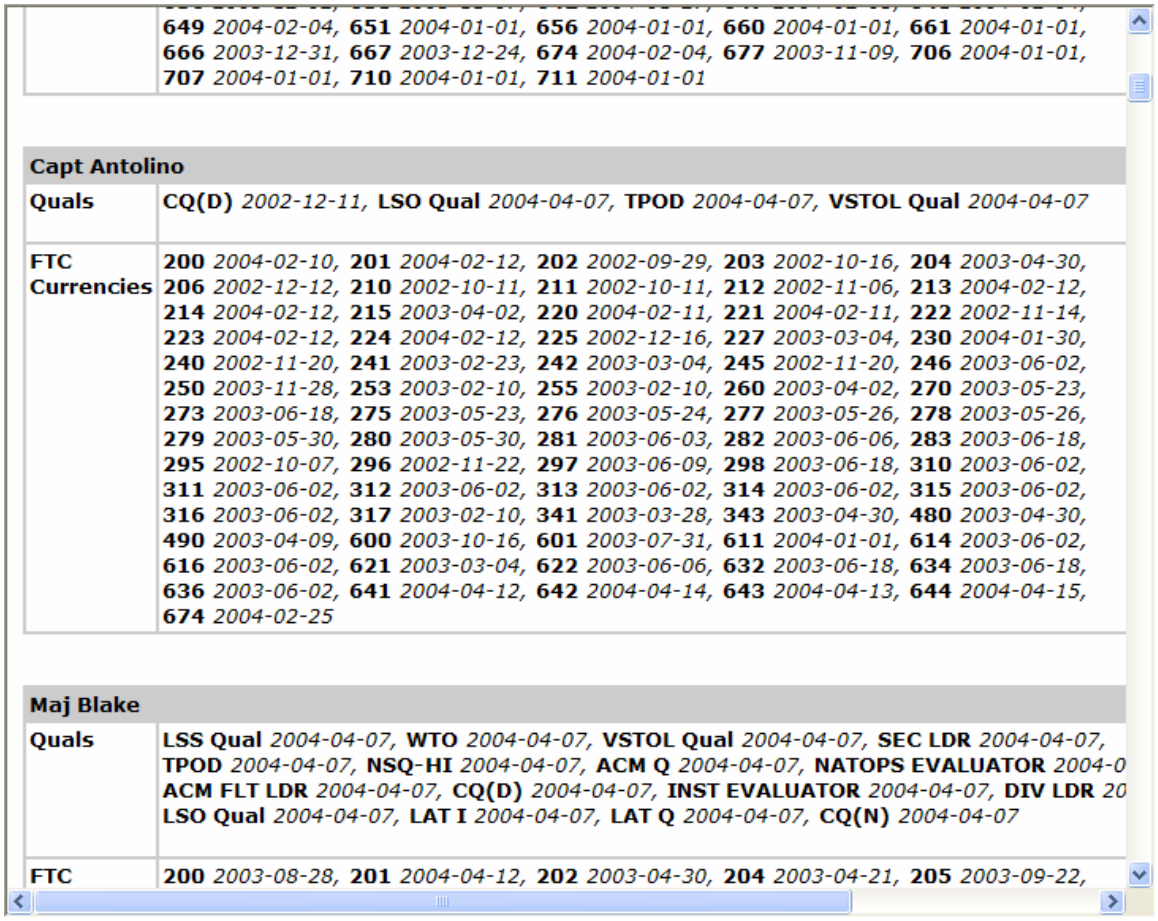
One of the lessons learned in the logistics challenge problem is that it is very difficult for users to specify an objective function that captures all the issues they care about. We found that it was only after seeing a solution that users would think about trade-offs that would be impractical to specify in advance (e.g., I'll accept having Jones fly 8 rather than 5 sorties if that is the only way I can get Smith's night systems qualification done).

2.2.3 Resources

The following is a list of the resources involved in producing operations flight schedules.

Pilots: A pilot's ability to participate in a task depends on the qualifications that the pilot has acquired. The set of qualifications that pilots have changes dynamically: as they participate in tasks they acquire new qualifications and as time goes on they may lose qualifications if they don't participate in tasks that refresh them. A typical squadron has 24 pilots.

The information about pilots is read in from a legacy database that the Marines use daily to update the pilot currencies after each day of flights. The information that SNAP uses includes the qualifications that each pilot has, and the last performance date of each training code. Figure 5 shows an excerpt of a report that SNAP produces after it reads the data from the legacy database.



	649 *2004-02-04*, **651** *2004-01-01*, **656** *2004-01-01*, **660** *2004-01-01*, **661** *2004-01-01*, **666** *2003-12-31*, **667** *2003-12-24*, **674** *2004-02-04*, **677** *2003-11-09*, **706** *2004-01-01*, **707** *2004-01-01*, **710** *2004-01-01*, **711** *2004-01-01*

Capt Antolino	
Quals	**CQ(D)** *2002-12-11*, **LSO Qual** *2004-04-07*, **TPOD** *2004-04-07*, **VSTOL Qual** *2004-04-07*
FTC Currencies	**200** *2004-02-10*, **201** *2004-02-12*, **202** *2002-09-29*, **203** *2002-10-16*, **204** *2003-04-30*, **206** *2002-12-12*, **210** *2002-10-11*, **211** *2002-10-11*, **212** *2002-11-06*, **213** *2004-02-12*, **214** *2004-02-12*, **215** *2003-04-02*, **220** *2004-02-11*, **221** *2004-02-11*, **222** *2002-11-14*, **223** *2004-02-12*, **224** *2004-02-12*, **225** *2002-12-16*, **227** *2003-03-04*, **230** *2004-01-30*, **240** *2002-11-20*, **241** *2003-02-23*, **242** *2003-03-04*, **245** *2002-11-20*, **246** *2003-06-02*, **250** *2003-11-28*, **253** *2003-02-10*, **255** *2003-02-10*, **260** *2003-04-02*, **270** *2003-05-23*, **273** *2003-06-18*, **275** *2003-05-23*, **276** *2003-05-24*, **277** *2003-05-26*, **278** *2003-05-26*, **279** *2003-05-30*, **280** *2003-05-30*, **281** *2003-06-03*, **282** *2003-06-06*, **283** *2003-06-18*, **295** *2002-10-07*, **296** *2002-11-22*, **297** *2003-06-09*, **298** *2003-06-18*, **310** *2003-06-02*, **311** *2003-06-02*, **312** *2003-06-02*, **313** *2003-06-02*, **314** *2003-06-02*, **315** *2003-06-02*, **316** *2003-06-02*, **317** *2003-02-10*, **341** *2003-03-28*, **343** *2003-04-30*, **480** *2003-04-30*, **490** *2003-04-09*, **600** *2003-10-16*, **601** *2003-07-31*, **611** *2004-01-01*, **614** *2003-06-02*, **616** *2003-06-02*, **621** *2003-03-04*, **622** *2003-06-06*, **632** *2003-06-18*, **634** *2003-06-18*, **636** *2003-06-02*, **641** *2004-04-12*, **642** *2004-04-14*, **643** *2004-04-13*, **644** *2004-04-15*, **674** *2004-02-25*

Maj Blake	
Quals	**LSS Qual** *2004-04-07*, **WTO** *2004-04-07*, **VSTOL Qual** *2004-04-07*, **SEC LDR** *2004-04-07*, **TPOD** *2004-04-07*, **NSQ-HI** *2004-04-07*, **ACM Q** *2004-04-07*, **NATOPS EVALUATOR** *2004-0* **ACM FLT LDR** *2004-04-07*, **CQ(D)** *2004-04-07*, **INST EVALUATOR** *2004-04-07*, **DIV LDR** *20* **LSO Qual** *2004-04-07*, **LAT I** *2004-04-07*, **LAT Q** *2004-04-07*, **CQ(N)** *2004-04-07*
FTC	**200** *2003-08-28*, **201** *2004-04-12*, **202** *2003-04-30*, **204** *2003-04-21*, **205** *2003-09-22*,

Figure 5: Pilot currency information extracted from a legacy database

Figure 6 shows a screen shot for viewing and specifying pilot information. The screen shows a timeline of activities for each pilot, including times of unavailability or SNIVELs (shown in tan), goals (shown in blue) and flights (shown in green). These

screens allow operators to specify information that is not currently captured in the legacy database.

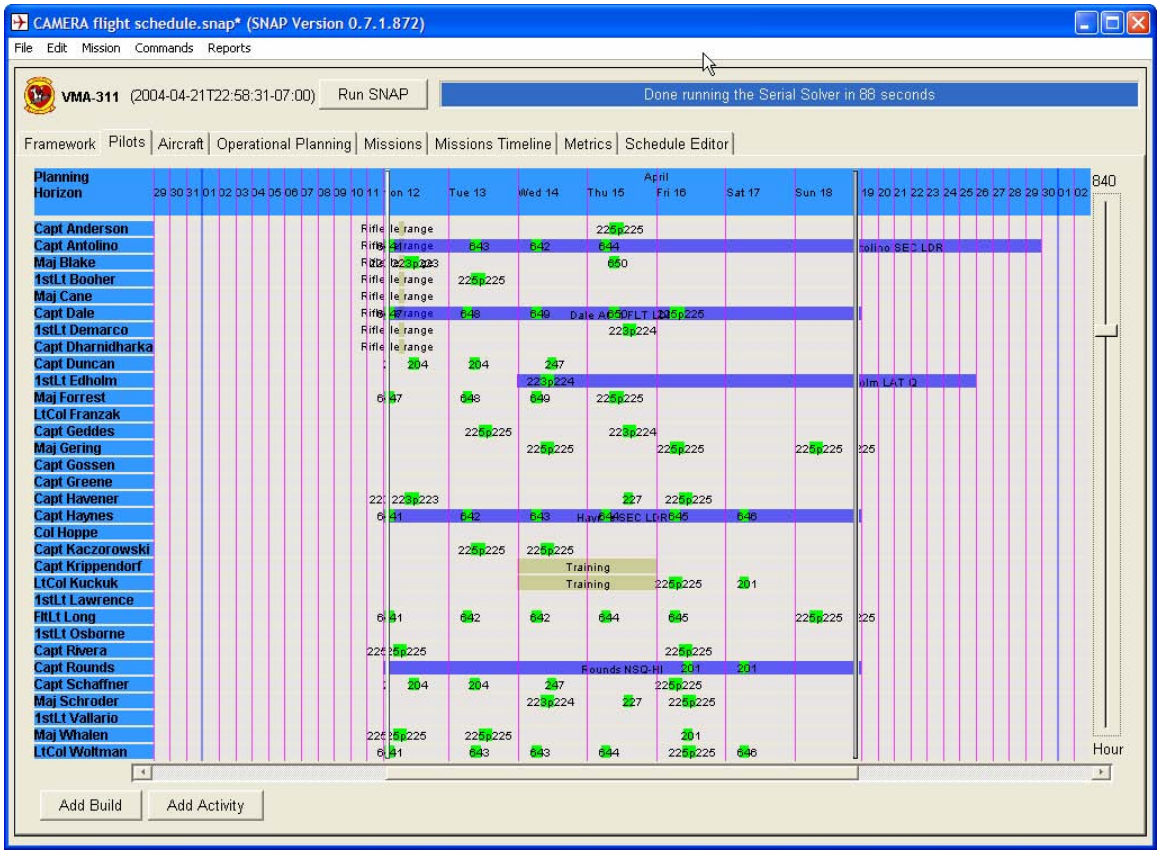


Figure 6: Viewing pilot schedule information in a time-line display

Aircraft: There are two types of aircraft, radar and night, and they can be used only for specific mission codes. A typical squadron has 16 aircraft.

Ranges: The places where missions are performed (e.g., dropping bombs, participating in air-to-air combat training). Different ranges are suitable only for specific mission codes. Suitability is defined using a preference ranking (e.g., large ranges are better for air-to-air combat). There are about 30 different ranges available to a squadron, but availability is very limited.

Simulators: Training on simulators is often required prior to actual flights. Different types of simulators are available for different training codes.

Sun light and moon light: Some mission codes can be performed at any time of the day, some can be performed only during the day and some during the night. In addition, some of the night codes require a certain amount of moon light and others require very low levels of light. Light level imposes interesting constraints because missing a moon cycle for performing a certain mission code may mean that the mission can only be performed several weeks later.

Sortie cycles: Conceptual resources that limit the number of flights to be performed during specific periods of a day. For example, 4 flights in the morning and 4 flights at night.

Figure 7 shows the Framework screen with summary information about ranges, simulators, sun light and moon light and sortie cycles. The sun light and moon light is shown in the Fly Days row. The green rectangles correspond to the hours of the day when flight operations will be conducted. The dark gray areas correspond to periods of darkness (less than 0.0022 LUX) and light gray areas correspond to periods of appropriate moon light (more than 0.0022 LUX) – in Figure 7 moon light is only available early Monday morning, which means that certain training codes can only be practiced during about a half hour period during the whole week.

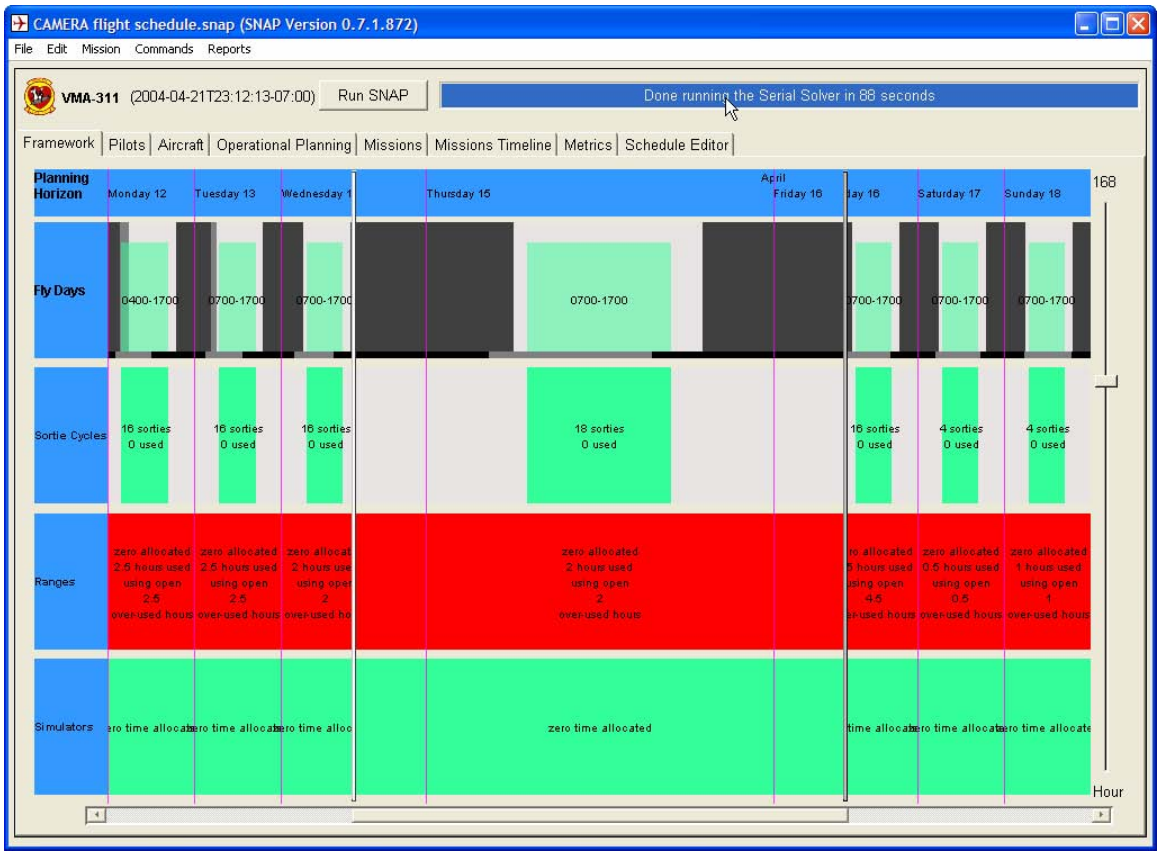


Figure 7: The overview screen for lighting, sortie cycles, and resources

2.2.4 Tasks

The tasks specify the activities that appear in the schedule. There are 3 types of tasks: on-base activities, simulator training and flights. On-base activities such as serving as operations duty officer involves booking a pilot qualified for that activity for a specific period of time. Simulator training involves booking a simulator, the pilot serving as the trainee and an instructor pilot. Flights are the most complex tasks, because they have rich

internal structure and involve all the resources.

The structure of a flight is defined in terms of the number of aircraft that participate in the flight, and in terms of a number of flight segments that specify at what points during the flight the different resources are needed.

- **Number of aircraft**: Although in general flights may consist of an arbitrary number of aircraft, the challenge problem focused on flights consisting of up to 4 aircraft. Most flights consist of two aircraft, and are called sections.
- **Segments**: The segments of a flight specify what is happening during the flight. A flight consists of a briefing segment, a number of flight segments, an optional stop for refueling followed by more flight segments, and finally a debriefing segment. Flights typically consists of 3 flight segments, one to go to the range, one to perform a mission code at a range, and one to come back from the range. Segments have a specified duration.

The structure of a task is shown in Figure 8, which shows the segments and the resources that participate in a task. The segments are shown as a time line at the top of the image. Each segment has a name and duration. For example, the first segment is the "brief" segment during which the pilots attend a briefing on the mission they are about to perform. Its duration is 120 minutes. Each row in the image represents a resource that participates in the task. The first two rows labeled "Lead" and "Wing" represent the pilots. The green bars represent the segments during which resources are needed. For example, the pilots are needed during all segments, whereas the aircraft are needed only after the briefing segment.

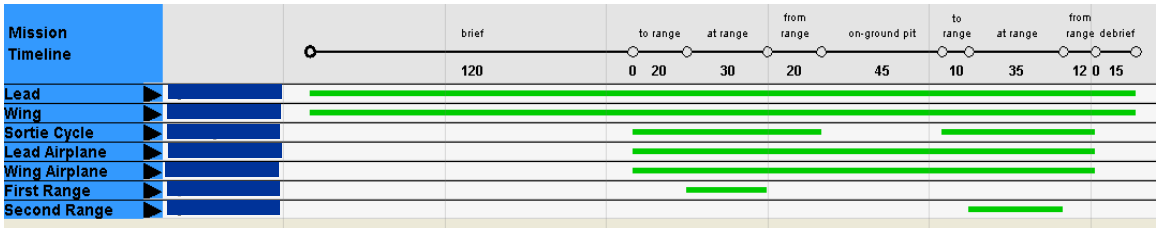


Figure 8: The interactive time-line display for fine-tuning mission segments

Resources participate in flights as follows:

- **Pilots**: Pilots are needed during all segments of a flight, starting with the briefing and ending with the debriefing. For each pilot, the segments are tagged with the mission code that the pilot will be performing during that segment. For example, the pilot in position 1 will be performing mission code 237 in the at range segment before the refueling stop, and participating in mission code 280 in the at range segment after the refueling. The pilot in position 2 may be performing different mission codes, and so on.

  In order to assign a pilot to a given position, he or she must have the minimum qualifications to perform the mission codes defined in all segments for that position. In addition, if a pilot is only minimally qualified to perform the mission codes in a position, one of the pilots in the other position must posses additional qualifications. In addition, one or several of the pilots in the flight must also posses certain

leadership qualifications (e.g., section leader). Light restrictions may impose further constraints on the qualifications because a pilot may be fully qualified to fly a mission in high light, but only minimally qualified to fly it in low light. This means that selection of pilots and selection of the time to fly a mission cannot be done independently.

- **Aircraft**: Aircraft are needed for all segments except for the briefing and debriefing. The challenge problem focused on scheduling of Harrier jet operations (AV8B), which are single seat aircraft.
- **Ranges**: Ranges are needed only for the segments where mission codes are being performed. The range must be suitable to perform all the mission codes listed in all positions during a given segment.
- **Sun light and moon light**: The light requirements for all mission codes must be satisfied. This sometimes creates tricky situations where the mission codes before a refueling stop are daylight codes and the ones after are night-time codes. This means that flight can only be scheduled around the sunset of days when moon conditions are appropriate.
- **Sortie cycles**: The sortie cycles are needed for all segments except for the briefing and debriefing.

2.2.5 Inter-Task Constraints

The previous section discussed the constraints that specify the required relationships between the resources that participate in a task. There are additional constraints that define relationships between tasks.

- **Pilot qualifications**: As mentioned before, pilots can gain qualifications after participating in certain tasks, and can lose qualifications if they don't perform certain tasks within certain periods. This means that in order to select pilots for a task it is necessary to consider potential selections for other tasks. For example, if Smith has a goal to fly mission code 281, but he has not flown 280, then it is necessary to select Smith in a task that has mission code 280, and that task must be scheduled prior to Smith's 281 task.
- **Crew rest and crew day**: There are many crew day and crew rest constraints. Pilots must fly less than a certain number of sorties and flight hours per day (the limits are different if they fly night missions). In addition, pilots must not be at base more than 10 hours every day, they must rest at least 8 hours between days, and the sleep pattern must not slide more than 2 hours per day (all the numbers are parameters that users can set). These constraints tie together the tasks in a very perverse way. For example, selecting Smith to fly early in the morning on Monday may result in him not being able to fly a night code on Wednesday, which may result in him not acquiring a qualification he needs on Thursday.
- **Location**: Flights may end in a different location from where they started. This means that the pilots and aircraft that participate in the flight will be at a new location. The scheduler must ensure that critical resources are scheduled in new tasks that take them to other locations where they are needed at critical times.

2.3 Producing Schedules

The previous section introduced the operations side of the logistics challenge problem by describing the elements that define the problem to be solved: the goals, objectives, resources, tasks and constraints. This section describes how the system solves the problem from the user's point of view. The next section describes the under-the-hood view of the system.

From the user's point of view, producing a schedule involves the following steps:

1. **Specify the problem**. In order to specify an operations problem users must specify the goals they want to achieve, and the resources available. They can ask the system to automatically generate the tasks to achieve the goals, and they have the freedom to add additional tasks, edit the tasks generated by the system, or delete generated tasks.
2. **Run the scheduler**. Once users specify the tasks they want to have on the schedule, they need to run the scheduler to have the system assign resources to the tasks and have the system determine the times when the tasks can be performed.

Figure 9 shows the screen where users manage the four different types of goals they can enter in the system (Wing FRAGs, Pilot Builds, Squadron Focus and Committed Events). The example screen shows that the user has entered 5 pilot build goals. The user can ask the system to automatically generate the tasks to achieve these goals by clicking on the Create Missions button at the lower right corner of their screen.

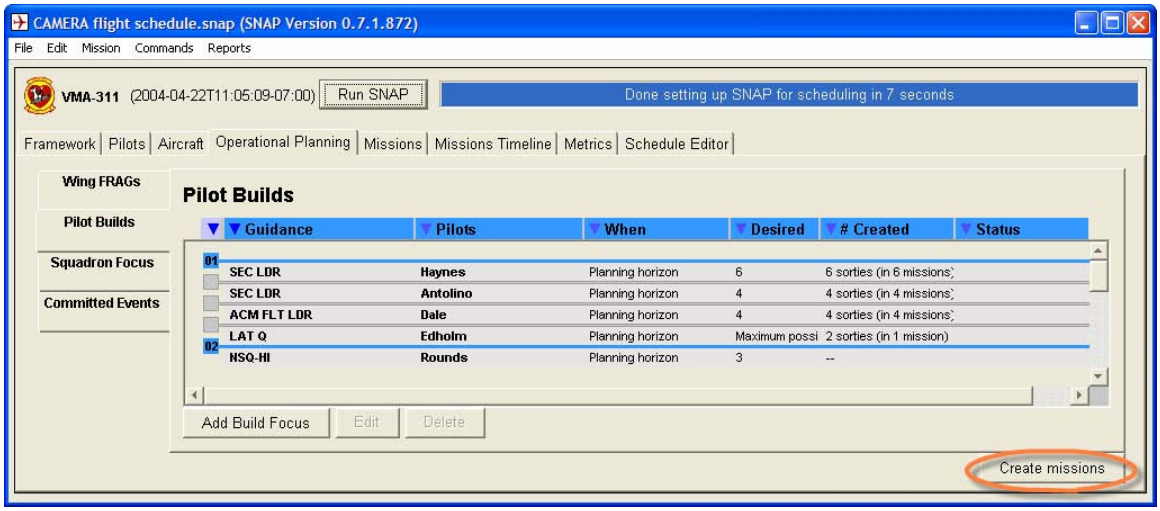


Figure 9: Managing high-level operational planning goals

Figure 10 shows the results after clicking on the Create Missions button. The tasks with a little yellow light-bulb icon are the ones that the system generated automatically in order to satisfy the goals. The other tasks are additional ones that users added to the schedule.

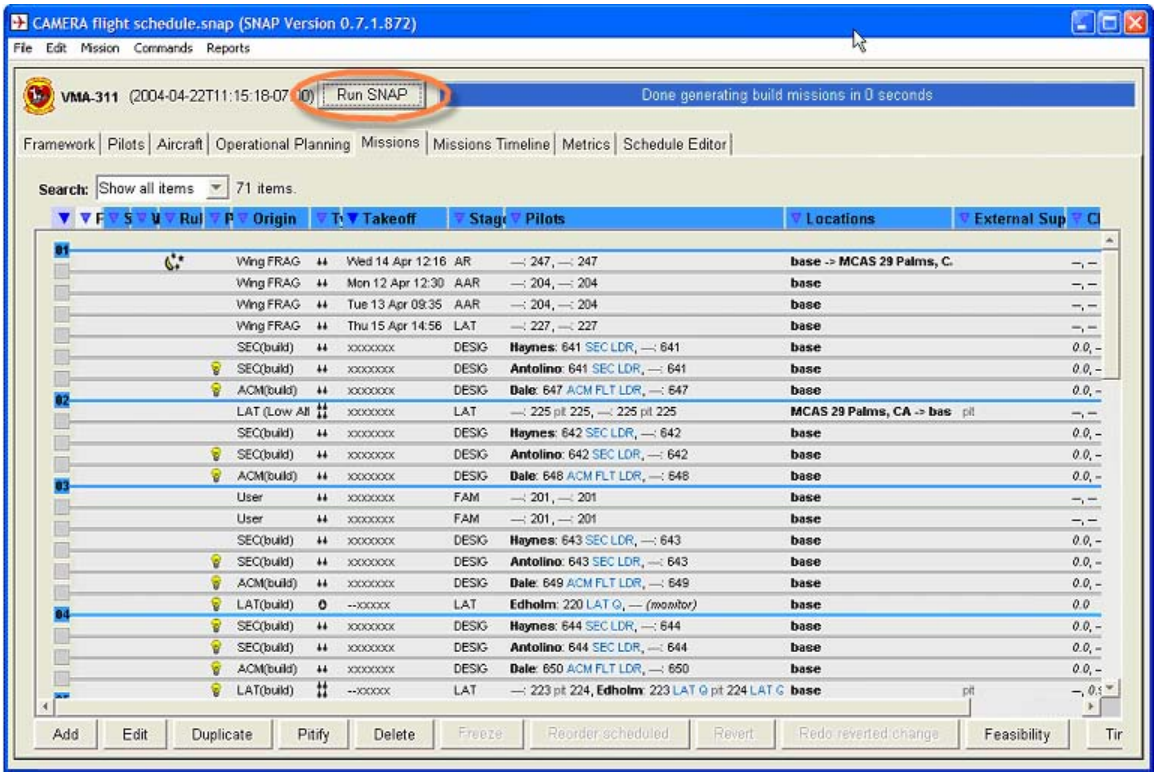


Figure 10: CAMERA-generated missions for the high-level goals

Once users are satisfied with the list of tasks, they have completed step 1 (Specify the problem) and are ready to perform step 2 (Run the scheduler). To run the scheduler, they click on the "Run SNAP" button at the top of the screen.

Figure 11 shows the results of running the scheduler. Scheduled tasks are marked with a green icon, whereas tasks that the system was unable to schedule are highlighted with a red X. Also, the resources and times that the system assigned to the tasks are shown in green. Users are free to override any system decision by selecting a task and invoking the task editor.

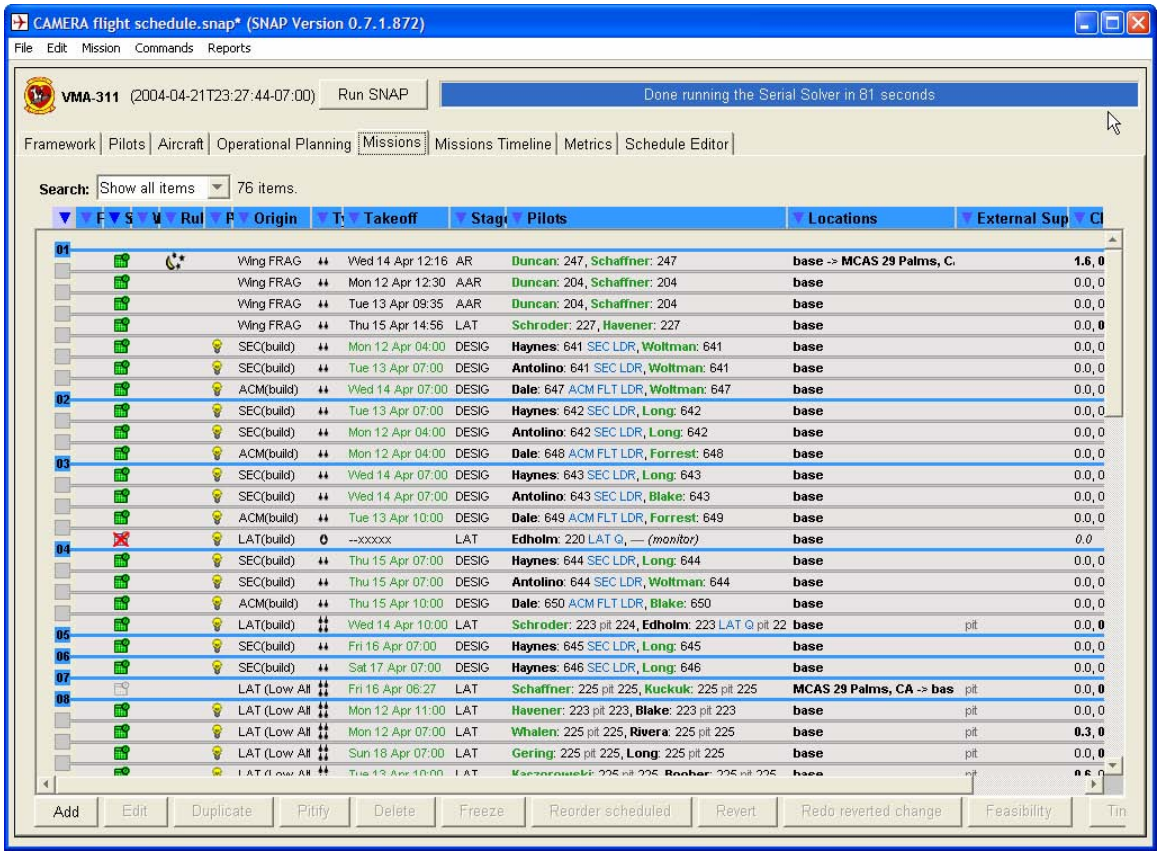


Figure 11: The resulting overall schedule (system choices are in green)

When a task fails to schedule, the user has to go back to step 1 (Specify the problem) to modify the problem so that the solution that the system can produce is satisfactory for the user. We call this iterative step problem reformulation.

There are typically many ways to reformulate a problem to achieve a satisfactory solution. The choice depends on the trade-offs that users are willing to make. Users can make more resources available, they can override certain constraints that prevent a task from scheduling (e.g., in the previous image the user waived the light constraints on the first task, as indicated by the moon and stars icon), they can change priorities, they can scale back their goals, etc. The system provides extensive editing capabilities that allow users substantial freedom in reformulating their problem.

When a task fails to schedule, the user can invoke the task feasibility display to understand why the task failed to schedule and what could be done to allow it to schedule. Figure 12 shows the feasibility display. It contains a row for each resource requirement of a task and for the constraints that it must satisfy. The planning horizon for the schedule is shown horizontally. Each point on the horizontal axis represents a potential start time (e.g., take-off time of a flight) for a task. The green bars in a row represent possible task start times when resources listed in that row and the rows above

are available. A red bar appears in a row when that row is the first row that causes some possible start times to be eliminated. The blue bars represent the start times when a particular resource is available. For example, the first row has uninterrupted blue and green bars. This means that the Mission Specified Time does not restrict the start time of the task at all. In other words, the user left the start time open. The second row corresponding to Fly Day Times has smaller blue bars. This is because the Fly Day specifies the times of day when flight operations are to be conducted and thus only start times that allow a task to start and end within those hours are allowed. The red bars in the Fly Day Times indicate possible start times that got eliminated because of the Fly Day restrictions. Similarly, going down the display it is easy to see that no pilot was available to fill the Lead position of the task until sometime late on Thursday. The Lead eliminated all possible start times on Monday through Thursday, and allows start times only on Friday, Saturday or Sunday. Looking further down in the display one can see that the task did not get scheduled because no appropriate ranges could be found (see last row).

The feasibility display is very powerful. For example, one can see that in order to schedule the task earlier during the week several things need to happen (in addition to making an appropriate range available). Wednesday is bad because no appropriate pilots can be found to fill the Lead or Wing positions. Tuesday is a good possibility because only the Lead is missing. Monday would work too, but might be harder to pull off because Wing pilots are only available for part of the day.

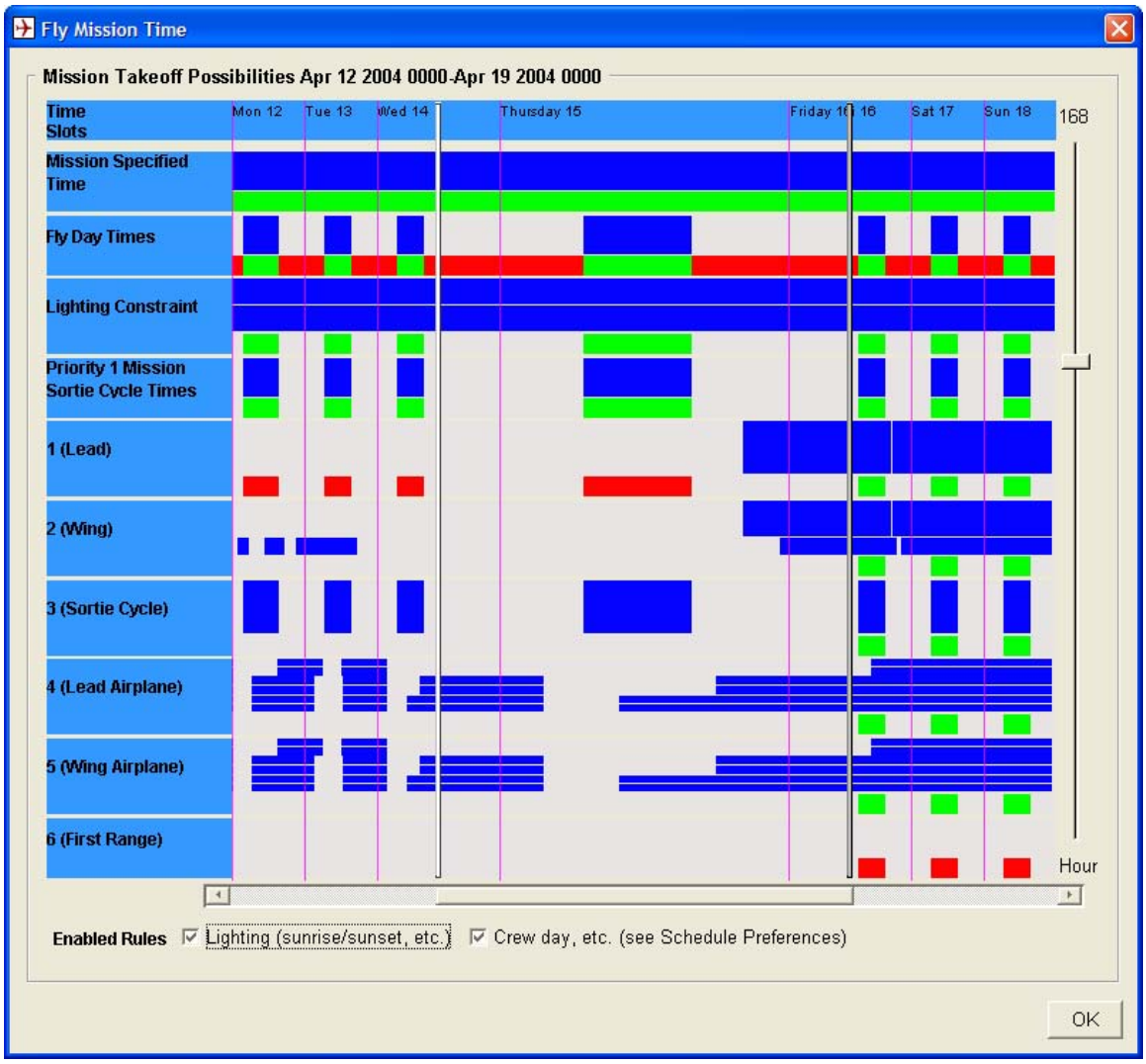


Figure 12: Understanding scheduling possibilities via the feasibility display

In practice, producing a satisfactory schedule often involves several reformulations. 10 reformulations for a weekly schedule is typical. Given that the scheduler can compute a weekly schedule in about 1 minute, the process can be completed in 15 to 20 minutes (it takes about 6 hours to complete a schedule without the system).

Once all tasks are scheduled as desired, the user can use the Schedule Editor to fill in the details of each task (ordnance, external support, remarks, etc). Figure 13 is a screen shot of this editor.

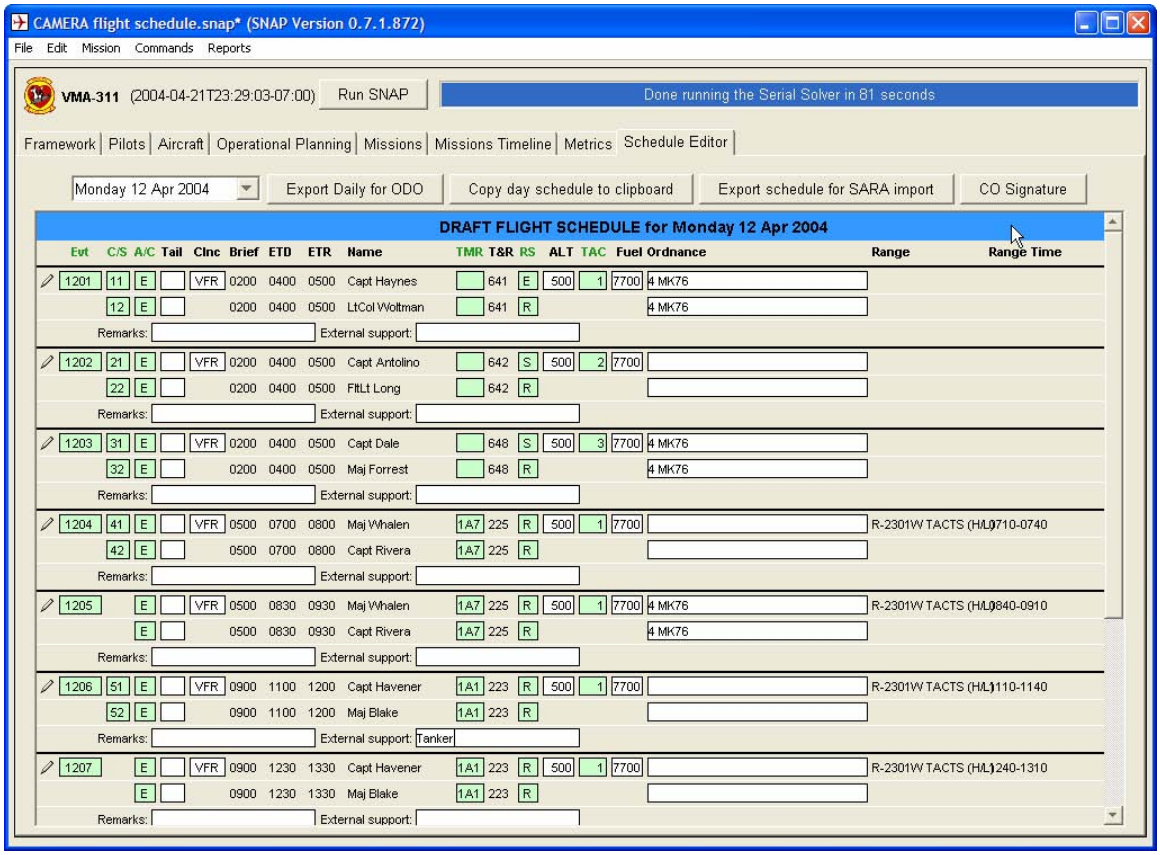
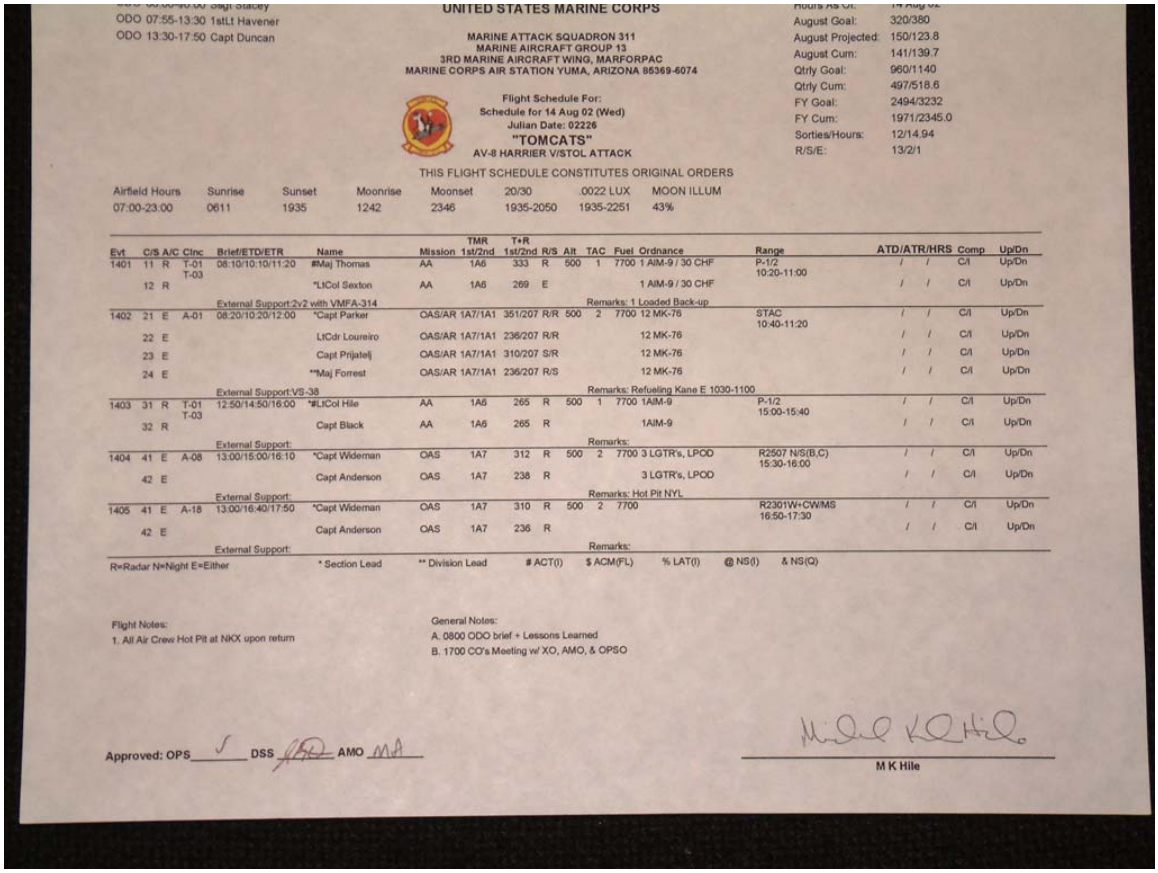


Figure 13: Fine-tuning details of sorties

Finally, users can publish the schedule on a Web server and print it for official signature. Figure 14 shows a photo of the first schedule produced with the system that was signed by a squadron commander and executed. The date on the schedule is August 14, 2002.



ODO 07:55-13:30 1stLt Havener
ODO 13:30-17:50 Capt Duncan

UNITED STATES MARINE CORPS

MARINE ATTACK SQUADRON 311
MARINE AIRCRAFT GROUP 13
3RD MARINE AIRCRAFT WING, MARFORPAC
MARINE CORPS AIR STATION YUMA, ARIZONA 85369-6074

Flight Schedule For:
Schedule for 14 Aug 02 (Wed)
Julian Date: 02226
"TOMCATS"
AV-8 HARRIER V/STOL ATTACK

August Goal: 320/380
August Projected: 150/123.8
August Cum: 141/139.7
Qtrly Goal: 960/1140
Qtrly Cum: 497/518.6
FY Goal: 2494/3232
FY Cum: 1971/2345.0
Sorties/Hours: 12/14.94
R/S/E: 13/2/1

THIS FLIGHT SCHEDULE CONSTITUTES ORIGINAL ORDERS

Airfield Hours	Sunrise	Sunset	Moonrise	Moonset	20/30	.0022 LUX	MOON ILLUM
07:00-23:00	0611	1935	1242	2346	1935-2050	1935-2251	43%

Evt	C/S	A/C	Clnc	Brief/ETD/ETR	Name	Mission	TMR 1st/2nd	T+R 1st/2nd	R/S	Alt	TAC	Fuel	Ordnance	Range	ATD/ATR/HRS	Comp	Up/Dn
1401	11	R	T-01 T-03	08:10/10:10/11:20	#Maj Thomas	AA	1A6	333	R	500	1	7700	1 AIM-9 / 30 CHF	P-1/2 10:20-11:00	/ /	C/I	Up/Dn
	12	R			*LtCol Sexton	AA	1A6	269	E				1 AIM-9 / 30 CHF		/ /	C/I	Up/Dn
				External Support:2v2 with VMFA-314									Remarks: 1 Loaded Back-up				
1402	21	E	A-01	08:20/10:20/12:00	*Capt Parker	OAS/AR	1A7/1A1	351/207	R/R	500	2	7700	12 MK-76	STAC 10:40-11:20	/ /	C/I	Up/Dn
	22	E			LtCdr Loureiro	OAS/AR	1A7/1A1	236/207	R/R				12 MK-76		/ /	C/I	Up/Dn
	23	E			Capt Prijatelj	OAS/AR	1A7/1A1	310/207	S/R				12 MK-76		/ /	C/I	Up/Dn
	24	E			**Maj Forrest	OAS/AR	1A7/1A1	236/207	R/S				12 MK-76		/ /	C/I	Up/Dn
				External Support:VS-38									Remarks: Refueling Kane E 1030-1100				
1403	31	R	T-01 T-03	12:50/14:50/16:00	*#LtCol Hile	AA	1A6	265	R	500	1	7700	1AIM-9	P-1/2 15:00-15:40	/ /	C/I	Up/Dn
	32	R			Capt Black	AA	1A6	265	R				1AIM-9		/ /	C/I	Up/Dn
				External Support:									Remarks:				
1404	41	E	A-08	13:00/15:00/16:10	*Capt Wideman	OAS	1A7	312	R	500	2	7700	3 LGTR's, LPOD	R2507 N/S(B,C) 15:30-16:00	/ /	C/I	Up/Dn
	42	E			Capt Anderson	OAS	1A7	238	R				3 LGTR's, LPOD		/ /	C/I	Up/Dn
				External Support:									Remarks: Hot Pit NYL				
1405	41	E	A-18	13:00/16:40/17:50	*Capt Wideman	OAS	1A7	310	R	500	2	7700		R2301W+CW/MS 16:50-17:30	/ /	C/I	Up/Dn
	42	E			Capt Anderson	OAS	1A7	236	R						/ /	C/I	Up/Dn
				External Support:									Remarks:				

R=Radar N=Night E=Either * Section Lead ** Division Lead # ACT(I) $ ACM(FL) % LAT(I) @ NS(I) & NS(Q)

Flight Notes:
1. All Air Crew Hot Pit at NKX upon return

General Notes:
A. 0800 ODO brief + Lessons Learned
B. 1700 CO's Meeting w/ XO, AMO, & OPSO

Approved: OPS √ DSS AMO

M K Hile

Figure 14: The first CAMERA-produced daily schedule as signed and flown

3 Technical Achievements and Insights

The CAMERA project produced the following technical achievements:

1. Open architecture for hybrid solvers
2. Integration of planning and scheduling
3. Making solutions understandable to end users
4. Handling of complex, ad-hoc constraints present in real world applications
5. Solving computationally intractable problems via kernel sub-problems
6. System of system coordination

The following sub-sections describe each of these achievements in detail.

3.1 Open Architecture for Hybrid Solvers

In recent years work on problem solvers has been able to produce high quality *generic* problem solvers that can solve large computationally difficult problems in diverse areas

such as scheduling, planning, circuit verification and Very Large Scale Integrated (VLSI circuit design. The input to these solvers is a standard, domain-independent representation such as mixed integer linear programming, 0/1 integer programming, or Boolean satisfiability and constraint satisfaction. To use such solvers, a problem is first encoded in one of these formalisms, then an off-the-shelf solver is applied to the encoded representation, and finally the solution is decoded to produce a solution to the original problem.

The CAMERA work made it clear that military problems such as scheduling flight operations are so complex that they cannot be effectively solved using any single off-the-shelf solvers. Military problems contain a diverse set of features and constraints. The off-the-shelf solvers can effectively address some of the features and constraints, but not others.

The main technical achievement of the CAMERA project is a hybrid solver architecture that allows several off-the-shelf solvers to be combined to solve a problem.

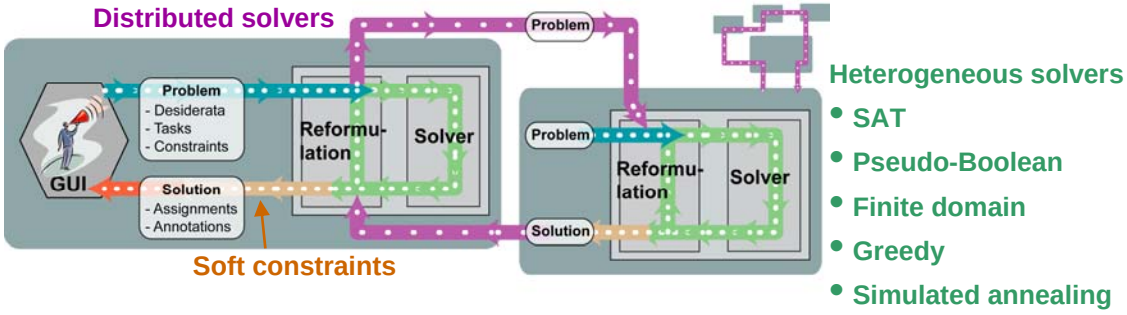


Figure 15: CAMERA's Hybrid Solver Architecture

Figure 15 shows the main elements of the CAMERA hybrid solver architecture. Rather than encoding a problem for a solver directly as traditionally done, in the CAMERA architecture the problem is first reformulated to extract a sub-problem that can be effectively solved using a specific solver. The solution from that solver is translated into soft constraints, and those soft constraints are added to the original formulation of the problem. The augmented problem is then solved using another solver. The key insight is that those soft constraints allow the second solver to be more effective because they provide information that allow the solver to reduce the search space for solutions. In essence, the soft constraints tell other solvers where to look for solutions. The reason to use soft constraints (i.e., constraints that the system is encouraged but not required to satisfy) rather than hard constraints is that the reformulation step may ignore certain features of the problem so that the first solver produces a solution that is not quite correct. Because the soft constraints can be violated, the second and subsequent solvers are able to recover from imperfect reformulations.

In the CAMERA project we developed one solver, the greedy solver. Other solvers were either off-the-shelf, or developed by other ANTS contractors:

Greedy solver: developed by the University of Southern California's Information Sciences Institute (USC/ISI). This is the production solver in use with the fielded version of the system. The greedy solver considers one task at a time, in priority order, and allocates a time and resources that maximize the objectives defined by the user. Once a task is allocated, the decisions are not revisited.

The benefits of the greedy solver are that it is fast (allocates about 1 task/second on a 2 GHz laptop), it respects all the domain constraints, and supports the feasibility display that allows users to understand why the system made the decisions it made.

The main weakness of the greedy solver is that it is prone to making sub-optimal solutions. When the greedy solver makes resource allocation decisions, those decisions are made based on the current task and the tasks it has already allocated. However the decisions don't take into consideration the tasks that have not yet been allocated. This means that the solver can select resources for the task at hand that prevent other tasks from being scheduled.

In practice, this weakness is not a significant problem. In fact, it is the greedy nature (one task at a time) of the greedy solver that makes the feasibility display possible (see section 2.3), and users find the feasibility display to be an indispensable part of the system.

Pseudo Boolean Solver: this solver consists of three parts: an encoder that encodes a operations scheduling problem into linear inequalities using variables that can take values 0 or 1, and an equation solver that solves the inequalities, and a decoder that translates the values of the variables back into the domain of flight scheduling.

The encoder and decoder parts of the Pseudo Boolean solver were developed by the Automated Tools To Evaluate Negotiation Difficulty (ATTEND) project (also at USC/ISI). We experimented with two solvers: one, the well known WALKSAT-OIP solver, which is based on local search methods, and OPARIS (developed by the University of Oregon – also an ANTS contractor).

The Pseudo-Boolean solver relied on the problem reformulation component of the architecture. Before encoding a problem in the Pseudo Boolean format it was first reformulated because the full complexity of a flight scheduling problem would produce encodings with millions of variables and equations which are unsolvable with any existing solver. The reformulation was parameterized and under user control:

- *Time resolution*: this reformulation changed the resolution of the problem from 1 minute to a coarser resolution (typically 30 minutes or 1 hour). When the resolution is coarser, the solver has fewer decisions to make. In the case of the Pseudo Boolean solver this manifests itself in significantly smaller encodings.
- *Task resource simplification*: this reformulation removes resources from consideration. In our experiments we found that some resources are widely available and they can be ignored when computing the start times of the tasks. Because they are plentiful, they can be allocated after the start times have been computed.
- *Constraint simplification*: this reformulation removes some constraints from consideration. Real world applications such as flight scheduling includes many complex legality constraints to enforce important, yet infrequent situations during

scheduling (e.g., pilots who have not flown for 15 days must perform a simulated mission before flying). These constraints can be ignored during the production of the schedule and later enforced by tweaking the resulting schedule. Ignoring these constraints during scheduling makes the problem significantly easier (less variables and constraints), and the hybrid architecture allows a simpler, local repair solver to enforce them.

The encodings of weekly schedules produce very large numbers of variables and inequalities (upwards of 50,000). The simplifications allow the Pseudo Boolean solver to produce weekly schedules at 1 hour resolution. The Pseudo Boolean solver considers all tasks simultaneously, and thus does not have the shortcomings of the greedy solver (making early decisions that prevent other tasks from being scheduled).

The Pseudo Boolean solver, as many other advanced solvers, has the weakness that the solutions it produces are hard to explain to users. If a task does not get scheduled, the solver does not provide any information about why this happened. Given that it has considered potentially millions of combinations, it is hard to extract a succinct reason why certain decisions were made. However, the hybrid architecture greatly ameliorates the problem because the greedy solver can be run after the Pseudo Boolean solver (or any other advanced solver), and it can produce a feasibility display that conveys a partial picture of why tasks were scheduled they way the were.

Simulated annealing solver: this solver developed by ALTARUM (ANTs contractor) is an iterative repair solver that works by starting with a partial solution to a scheduling problem and iteratively refining it. This solver was integrated into the CAMERA architecture, and we demonstrated how it can be used to effectively repair solutions after users make minor modifications to the inputs.

Market-based solver: this solver uses an auction based protocol for scheduling tasks. The basic idea is that tasks bid on resources, and the highest bidder wins the allocation of the resource. This solver was developed as part of an Air Force Office of Scientific Research grant, and also integrated into the architecture. This solver also offered useful schedule repair capabilities.

3.2 Integration of Planning and Scheduling

Another example of the complexities of real world applications is that the flight scheduling application is not a pure scheduling application. In scheduling applications the input is a set of tasks and resources, and the problem is to compute the start times and resource allocations for each task. The flight scheduling application had two features that bring planning aspects into the scheduling application, which greatly increase the complexity of the problem.

3.2.1 The "Create Tasks" Problem

The operations scheduling problem is a planning and scheduling problem. The goal statements (e.g., achieve night systems qualification for Smith) specify what needs to be done, whereas the tasks specify how it will be done. The challenge problem requires creating the tasks and scheduling them. For example, to achieve the night systems qualification for Smith, he must fly mission codes 280 and 281. This could be done with one task with a refueling stop such that 280 is done before the refueling and 281 done after. It could be done with two tasks without a refueling stop, and in fact, could be done in other kinds of tasks.

One simple approach to solve this problem is to have a "create tasks" phase that creates tasks to meet the goals, followed by a scheduling phase that schedules the tasks. Of course, the difficulty is that the scheduling problem could be made much easier if a different set of tasks had been generated. However, without the information gathered during scheduling it is not possible to determine which tasks make the scheduling problem easier.

Alternative approaches are to interleave task creation and scheduling, or to do both things at the same time. The difficulty here is that the search space becomes much larger because it converts a scheduling problem into a planning and scheduling problem.

The solution to the challenge problem used the first approach even though it sometimes led to unscheduled tasks and obviously sub-optimal solutions. However, the system gave users the ability to overcome the problem by allowing them to edit the tasks to restructure them to be more amenable to the scheduler. In typical schedules users only had to do this a few times.

The maintenance scheduling problem that Vanderbilt University worked on faced a similar problem. Given a flight schedule, the maintenance tool would assign aircraft to flights, making assignments to maximize the number of flights that could be supported. Once aircraft are assigned to flights, the tools can forecast the flight hours for each aircraft and create the tasks to perform maintenance on the aircraft. Again, the aircraft assignments were done without consideration of how it affects scheduling of maintenance on the aircraft. This sometimes resulted in unscheduled tasks or sub-optimal solutions. The solution here was also to let the user edit the aircraft assignments to make the maintenance scheduling easier.

3.2.2 The Enablement Problem

The enablement problem is similar to the "create tasks" problem in that it brings planning aspects into the scheduling problem. The enablement problem concerns qualifications that pilots gain by participating in tasks and qualifications they lose from not participating in certain tasks. We called it the enablement problem because participating in certain tasks enables pilots to participate in others. If participation in task T gives a pilot a qualification to participate in task S, then if the system wants to select that pilot for both S and T, then task S must be scheduled after task T. The difficulty is that the relationship between tasks S and T is dynamic in the sense that it depends on the

assignment of pilots to the tasks: if a pilot is already qualified to participate in both S and T, then those two tasks can be scheduled in any order.

The enablement problem is significantly easier than the "create tasks" problem because it does not change the structure of the tasks. The problem was solved in two different ways.

The Information Sciences Institute (ISI)/Cornell collaboration produced a solution based on a Pseudo-Boolean encoding of the problem where the system automatically figures out both pilot assignments and timing of tasks when it is optimizing the number of scheduled tasks.

The ISI team solved the problem in the greedy solver to provide a fast solution to the most common cases. The most common occurrences of the enablement problem are in the tasks generated to satisfy qualification build goal and in the tasks to achieve new core competencies. In those tasks, one of the pilots is always known, i.e., the pilot assignment is constrained to have a single value. This allows the greedy scheduler to sort the tasks in enablement order so that if tasks T enables task S, the scheduler first schedules task T and when it schedules task S, it can select the start time of S to be after the end time of T. This simple approach covers the most common cases although it can lead to sub-optimal solutions. For example, when scheduling task S the greedy scheduler does not consider task T, so it doesn't know not to delay task S (e.g., to optimize another objective such as range preference) in a way that task T is squeezed out of the schedule.

3.3 Making Solutions Understandable To End Users

Understandability refers to the users' ability to understand why tasks were scheduled the way they were, and most importantly, why an allocation they had in mind had not been done. In schedules with short planning horizons users would often edit the tasks to add specific constraints such as specifying both pilots. The additional constraints made the problem harder to solve, often resulting in tasks not being scheduled. In those cases, users wanted to know why their tasks had not been scheduled.

In general, such questions are very hard to answer succinctly because there are a large number of constraints that affect the allocation of a task, and the chains or reasoning can be long. The solution to this challenge takes advantage of the simplicity of the greedy solver, and the ability of the greedy solver to essentially hide from the user the sophistication of the pseudo-Boolean solver when using the 2-phase solver.

The why, and why not questions are answered using a feasibility display that shows all the possible times when the resources that can participate in a task are available. The display cleverly arranges the timeline for each resource and each constraint class so it is easy for the user to see which resource or which constraint prevents a task from getting scheduled during all times in the planning horizon. The users learned how to use this display, and were able to solve almost all scheduling questions using it.

The feasibility display relies on the sequential nature of the greedy solver in that the ability to schedule a task depends only on the tasks that have been scheduled so far. This means that the feasibility display does not work for solvers that do backtracking. The 2-

phase solver enabled a compromise in that the feasibility display is built from the results of the greedy solver which runs after the more sophisticated pseudo-Boolean solver. The result is that the feasibility display shows only one of the reasons why something didn't happen (based on the reasoning abilities of the greedy solver), but that is enough to let users fix problems. Without the feasibility display users were unable to fix scheduling problems, and thus were unable to use the system to solve practical problems.

3.4 Handling of Complex, Ad-Hoc Constraints Present In Real World Applications

The flight scheduling application featured many ad-hoc (application-specific) constraints that were difficult to represent in a generic way, and that were difficult to encode in the generic formalisms of the off-the-shelf solvers.

Most of these constraints capture safety and legality issues. The hybrid architecture provided an approach to deal with these constraints. The approach involved ignoring the constraints that are triggered infrequently during the early stages of scheduling, and enforcing them during the last stage of scheduling. This was done in the problem reformulation component of the architecture where problems were reformulated to ignore or approximate these constraints. This gave the advanced solvers more tractable problems. In the last phase of scheduling, the solution of the advanced solvers was converted into soft constraints and added to the original problem that contained the difficult to handle constraints. The greedy solver, with the help of the soft constraints avoided the usual pitfalls of making early incorrect decisions, and at the same time enforced all constraints, often repairing the solutions from the advanced solvers to meet the constraints that had been previously ignored.

3.4.1 The Crew Day and Crew Rest Problem

The crew day and crew rest constraints pertain to pilots, and essentially enforce that pilots get enough rest every day. They are parameterized with respect to the length of the work day and time of last landing, with respect to the number of hours of rest and the shift of sleep patterns from day to day.

Crew day and crew rest constraints cannot be ignored during the reformulation phase of problem solving because they affect every pilot allocation to a task.

In the greedy solver 50% of the time was spent evaluating the crew day and crew rest constraints in order to compute the time intervals when these constraints allowed pilots to fly. The problem is that the solver invokes this computation each time it considers a time point for a pilot because for each proposed time it must figure out if there is a way to arrange the work and rest patterns of the pilot to make them consistent with all other activities the pilot has in the schedule. The ISI team is considering less sophisticated algorithms that would miss some of the more creative solutions, but would be significantly faster.

In the pseudo-Boolean solver the number of variables and constraints devoted to encoding these constraints are 3 times more than the variables and constraints used to encode the rest of the problem.

In both solvers, the parts of the solver that address these constraints were written several times and significant development time was spent just on this aspect of the system. Even though crew day and crew rest are just one of the constraints in the logistics problem, it was important to address them thoroughly because these constraints are critical to safety. A general, fast solution to this type of constraint remains an important unsolved problem.

3.5 Solving Computationally Intractable Problems Via Kernel Sub-Problems

Our experiments with the hybrid solver architecture revealed that the large flight scheduling problems often contain a small kernel sub-problem that captures the essence of the original problem. Once the small kernel problem is solved, it is easy to find a solution to the much larger original problem.

Figure 16 illustrates the kernel sub-problem idea in the context of difficult 3-day surge scheduling problem. The problem is difficult because a surge schedule contains a large number of tasks, but not a larger number of resources so that finding a solution is significantly harder. For example, the greedy solver was able to schedule only 18 of the 21 tasks. The chart shows that if one desires a high fidelity solution (4 minute resolution), the Pseudo Boolean encoding would contain half a million variables and over a million constraints. This is too large for even the most sophisticated solvers. Problem reformulation can simplify the problem significantly. The chart shows that by reducing the time resolution the number of variables and constraints is reduced very rapidly. This moves the problem away from the region of problems that are too complex to solve. However, reducing the time resolution beyond a certain point produces a reformulation of the problem that is too inaccurate. This means that once the reformulated problem is solved, the soft constraints generated from its solution are too weak to effectively guide other solvers towards a solution to the original problem. The result is that the later solvers are not better off, and in fact the soft constraints end up misguiding those solvers and making their job harder.

The lower curves in the chart show the complexity reductions achievable by ignoring resources and simplifying constraints. In the example shown, the best kernel problem involved changing the time resolution to 30 minutes and ignoring the range constraints. The blue arrow shows that the original problem could be simplified into a kernel problem that contained less than 30K variables. The Pseudo Boolean solver was able to solve the kernel problem in 3 minutes, and the greedy solver was able to use the soft constraints generated from that solution to solve the original problem in about 30 seconds, taking into account all constraints and resources, and producing a solution to a 1 minute resolution.

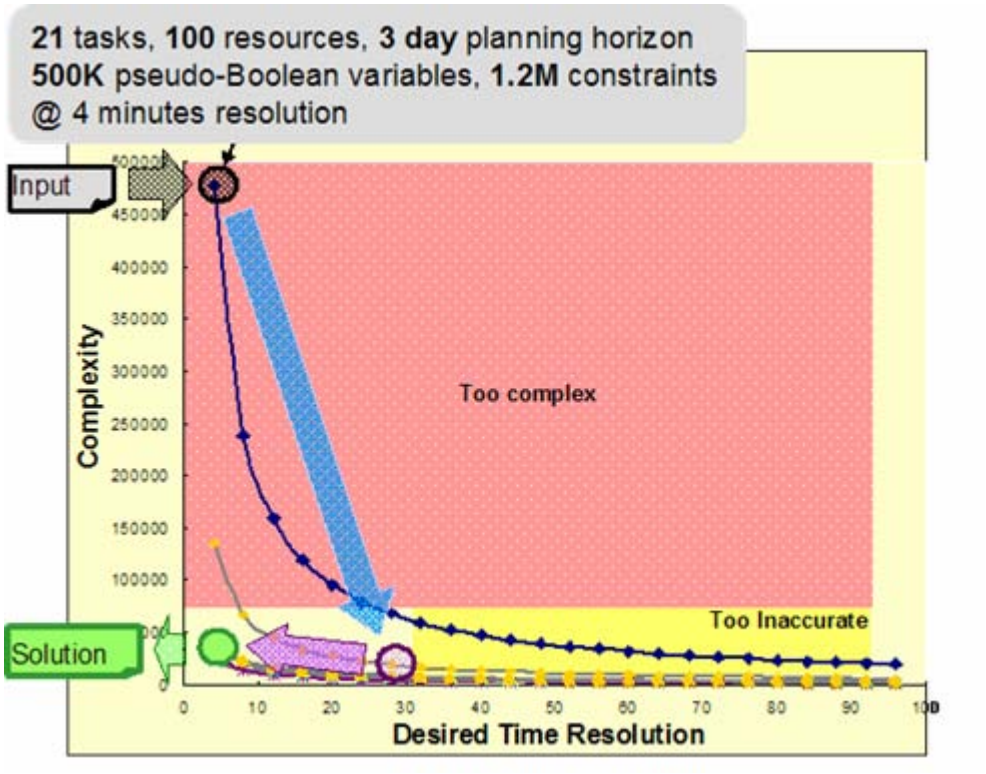


Figure 16: Reducing complexity by solving a kernel sub-problem

3.5.1 Short and Long Planning Horizons

The logistics challenge called for computing daily, weekly and monthly schedules, all with the same level of detail. The technical challenge was to produce scheduling algorithms that scale linearly with the size of the planning horizon. The greedy scheduler is almost linear with the size of the planning horizon because it schedules one task at a time and does not do backtracking. The pseudo-Boolean solver is not linear with the size of the planning horizon because its complexity is not linear with the number of tasks, and the number of tasks grows linearly with the size of the planning horizon. The greedy solver computed daily schedules in a few seconds, and monthly schedules in about 5 to 10 minutes.

Even though the 2-phase solver offers the opportunity to address long planning horizons by discretizing long planning horizons very coarsely (say to 6 hours), this was not tried because coarse discretizations should be conceptually different from fine-grained discretizations. For example, crew day and crew rest make no sense when discretized more coarsely than the amount of shift allowed between adjacent crew days (typically 2 hours). In that case, crew day and crew rest constraints should be replaced by capacity constraints, that for instance, limit pilots to fly twice a day. There was no time to implement such problem reformulations.

3.6 System of Systems Coordination

The flight scheduling problem is only one part of the problem of producing a viable flight schedule. The other part involves ensuring that aircraft maintenance can be arranged to support the flights in the schedule.

A CAMERA sister project at Vanderbilt University addressed the maintenance aspect of the problem. USC/ISI and Vanderbilt University collaborated to address the coordinated operations/maintenance problem. The issue is to compute a flight schedule and an aircraft availability forecast such that it supports the flight schedule. This comes about in the following way:

1. Operations uses an estimate of the available aircraft to produce a flight schedule.
2. Maintenance produces an aircraft availability forecast based on the flight schedule.
3. Operations verifies that the aircraft availability file supports the flight schedule, i.e., no changes to take-off times are needed.
4. If the aircraft availability forecast forces changes to the flight schedule then repeat the cycle, i.e., go to step 1 and use the new availability forecast as a revised estimate.

If operations makes any changes to the flight schedule, then the cycle must be repeated until it is verified that maintenance can support the flight schedule. If maintenance needs to change the forecast, the cycle must also be repeated to verify the new forecast supports the flight schedule.

As mentioned before, in the end the squadron is better off when operations produces more maintainable schedules. This means that the scheduling of flights is made sensitive to maintenance requirements. This defines a spectrum of possibilities: in one end of the spectrum (full sensitivity), the operations and maintenance problems are solved as a single larger problem; at the other end of the spectrum (no sensitivity), the problems are solved independently and only the results are exchanged between the parties. In the full sensitivity mode steps 1 through 4 wash out into a collection of constraints that tie the requirements together. In the no sensitivity mode the steps 1 through 4 are done with the results of the schedulers and repeated until convergence is reached. In the middle points of the spectrum operations and maintenance exchange additional information than just point solutions (e.g., ranges of take-off times) and repeat the cycles incrementally (e.g., for single aircraft rather than for all aircraft at a time).

During this project only the no-sensitivity mode was explored. The operations scheduler produces a fully detailed flight schedule. The maintenance scheduler uses the fully detailed flight schedule to produce a single aircraft availability forecast (i.e., with no what-ifs), and steps 1 through 4 were repeated until convergence was reached. Even though this was a very simple solution, the outcome was much better than the pre-existing way of doing business, which involved coordinating the schedules by talking on the phone.

3.7 MARBLES Market-Inspired Negotiation

We also investigated market-inspired negotiation algorithms in the context of an

idealized version of the flight scheduling problem. These algorithms are distributed in nature and can in principle also be applied to, say, robotic soccer players or Unmanned Combat Air Vehicles (UCAVs). These agents can act individually but are better off coordinating with their peers. A subset of this problem is distributed real-time resource allocation – deciding under time pressure which soccer player will take the final shot on the goal, or which UCAVs will neutralize a newly discovered enemy threat.
There is a spectrum of approaches for distributed real-time resource allocation, ranging from no communication at all (physics-based approach: agents observe each others' behavior but do not explicitly communicate, much like a wolf pack closing in on prey) to communication of the full rationale of behavior (argumentation approach: agents back up requests to others by an argument of why they should grant it).
In this continuum, our Marbles schemes and other "market-inspired" approaches fall in-between. Compared to a purely physics-based approach, they obviously use more messages yet also explore a more complex set of alternatives. Compared to the argumentation approach, they exchange messages of smaller complexity, yet the prices set by supply and demand can possibly communicate rationale in an alternative, more compact fashion, and potentially steer the group of agents to sensible behavior via "the invisible hand of the market".

3.7.1 External Marbles Scheme Properties

"Marbles" schemes[1] are a family of resource allocation algorithms that are characterized by the following properties:
Distributed. Each task only knows about its local requirements, and communicates with potential resources for those requirements exclusively through messages. Hence, each task and each resource can – but does not have to – be located on a different machine.
Cooperative. Marbles schemes are not designed to tolerate malicious participants, which distinguishes our research from work on e.g. electronic commerce and automated auctions; we believe that security against external attack of cooperative negotiation schemes is best located at a lower level (such as the message transport and encryption level). The cooperative nature of the negotiation also means that tasks participating in resource auctions can altruistically commit suicide by permanently withdrawing, and therefore lowering resource prices that possibly help others succeed. The distributed algorithms for concluding that tasks are unlikely to succeed further distinguishes our work from work on competitive auctions.
Adaptive. A Marbles scheme can adapt a current partial solution to a new situation rather than having to re-compute the new solution from scratch. This makes them applicable in cases where "the world can't stop while a solver computes a solution for everyone", that is, in cases where the time interval between situation changes is smaller than the total running time of a non-adaptive centralized solver.
Real-Time. The individual negotiation participants should be explicitly aware of time and adapt their behavior based on how much time is left.
Fault-Tolerant. A Marbles scheme should be robust against a set level of message loss, in

[1] The name is not an acronym, a team member likened the agent behavior to "kids trading marbles" in an early design discussion, and the name stuck.

the sense of being able to make statements like “given an average message delay of 2 seconds and a message loss rate of 5%, this negotiation has a 99% likelihood of concluding in less than 3 minutes”. Obviously, no message-based scheme can ever be robust in the sense of making a 100% real-time response guarantee if there is a non-zero chance of a message getting lost.

3.7.2 Internal Marbles Scheme Properties

We further characterize Marbles scheme by their “internal” properties; that term is accurate in the sense of being more linked to our approach than the above “external” ones. However, these choices do “shine through” to the user level, so the distinction between external and internal is not as sharp as it may sound.

Domain-based task valuation. Marbles schemes put a value on the execution of tasks that is quantitative and that has meaning to domain practitioners. The value of resources is exclusively derived from the value of the tasks they enable; they have no intrinsic domain value of their own.

Lack of inflation. We do not allow inflation (the artificial introduction of currency not backed up by domain value during negotiation) because the overall solution can otherwise not be verified in domain terms. For example, imagine that a negotiation scheme introduces inflation by increasing the value of tasks the longer they go unfilled during negotiation: consequently, the basis of the proposed overall solution cannot be analyzed by a domain expert without understanding the negotiation algorithm. Thus, the problem with mixing intrinsic task value and negotiation-scheme-dependent artificial “value” is that it would make the term “domain” currency meaningless.

Ever-fluctuating prices. In the prototypical open-outcry auction, participants bid until no one wants to bid higher, and the highest bidder then owns the resource from that point in time on. In contrast, Marbles schemes resources continually auction themselves - the auctions never “close”. That is, you can only be the current, not final, winner, of a resource -if the situation changes because, e.g., a new high-valued task appears you will lose it.

3.7.3 Formal Problem Statement

Below we introduce the minimalistic problem statement that our existing Marbles schemes operate on. In the future, we will continually expand the problem definition to, e.g., be able to shift tasks in time, to introduce a notion of equity in resource use, and so on, but the current problem already captures the essential challenge of distributed resource allocation. Note that none of our existing Marbles schemes presented below exhibits all of the desirable properties outlined in the Marbles Vision section; in particular, none of them can make real-time response or fault tolerance guarantees yet.

Problem

There is a collection of available resources that are characterized by a unique name (and nothing else). There is a collection of possible tasks that are worth a fixed domain value if they are executed. They need to acquire one resource for each of their requirements to be executed. Each resource can only be used for at most one task. Each task knows in

advance which resources are suitable for its requirements. Thus, we neglect a prior "resource discovery" phase. This problem is very complex if tasks have multiple requirements ("complementaries" exist, in economic jargon) - it would be trivial if each task had just a single requirement.

Solution

A solution consists of an assignment of resources to requirements such that every task has either none or all of its requirements filled. The quality of a solution is measured by the sum of the domain values of its satisfied tasks; a higher sum indicates a better solution.

3.7.4 Running Example

We will use the following example depicted in Figure 17 to explain how the various Marble scheme variants operate. There are four resources called A, B, C, and D. There are two tasks called Q and R of domain value 300 and 100, respectively. Each of the two tasks has two requirements that can be filled by the resources indicated with a triangle. This particular example was chosen because it is small yet leads to backtracking behavior if schemes assign resources to requirements from left to right (as they usually do). The optimal solution of domain value 400 is obvious (Q gets A and D, R gets B and C).

		A	B	C	D
Q (300)	1	▲	▲		▲
	2	▲			
R (100)	1	▲		▲	
	2		▲		

Figure 17: The running example problem

3.7.5 A Rough Taxonomy of Solvers

We will present a number of "solvers" – any piece of code that produces a solution given a problem in the above terms. Our research interest is exclusively in fully distributed resource allocation schemes, but we have also built a number of centralized solvers for comparison purposes. In addition, some of our Marbles variants have so far only addressed part of the challenge in a distributed way because we have not had the time to make them fully distributed.

All Marbles solvers fundamentally perform two tasks: assigning resources to the highest-bidding tasks ("allocation"), and eliminating tasks from competition ("elimination") because they drive up the prices for others without seeming to have a chance of obtaining all of their needed resources. Each of the variants indicates if it solves each phase in a distributed or centralized fashion.

Marbles2 [*allocation*: distributed, *elimination*: centralized]

The main inspiration behind this Marbles variant is that the cost of a resource should be defined by the value that the second-highest bidder places on it (the "displacement" or "opportunity" cost of the resource). Consequently, resources cost zero if no one else wants them.

Message Protocol
Task to resource: bid(amount), withdrawal(); resource to task: loss(), win(amount that can be lower than bid), priceChange(can be up or down but recipient is still winning).

The Running Example under Marbles2
In this variant of our Marbles schemes, tasks attempt to fill each requirement one at a time, bidding all of their available value to satisfy the next unfilled requirement.
1. Q simultaneously bids 300 on A, B, and D to satisfy its first requirement. R bids 100 on A and C.

		A	B	C	D
Q	1	▲300	▲300		▲300
(300)	2	▲			
R	1	▲100		▲100	
(100)	2		▲		

Figure 18: First stage of marbles2 solution to the problem

2. Q obtains A for 100 (the cost as a displacement cost is determined by the second highest bidder). It reacts by bidding 200 for B and D (because it has internally determined that it is better off by using A for its second requirement 2, and has already spent 100 of its 300 value for obtaining a resource.

		A	B	C	D
Q	1	▲100!	▲200		▲200
(300)	2	▲			
R	1	▲100		▲100	
(100)	2		▲		

Figure 19: Second stage of marbles2 solution to the problem

3. R wins C for 0 (as there are no competing bidders). It reacts by bidding 100 on B to obtain its second resource, and by completely withdrawing its bid for A (it already has a resource for its first requirement for free; otherwise it would have bid on A whatever it had to pay for C minus the minimum bid increment/decrement).
4. Q gets notified that the price of its A dropped to 0 (because all competition disappeared). It thus now increases its bids for B and D to 300. Exclamation marks indicate that the tasks are currently winning the resource.

		A	B	C	D
Q	1	▲0!	▲300		▲300
(300)	2	▲			

[2] Interestingly, we handle internal assignments by an internal use of the very same Marbles scheme where each requirement gets the same constant value to bid on resources won by the task, but we won't go into the details of that here.

R	1	▲100		▲0!	
(100)	2		▲		

Figure 20: Third stage of marbles2 solution to the problem

5. Q wins B for 100 (because that's R's bid). It is now satisfied, but bids 99 for D (a cheaper resource is always preferable) just in case.
6. Q wins D for 0 because no one else wants it. It withdraws its bid for B because nothing beats a free resource.
7. R gets notified that it is now the winner on B (also for 0). The scheme is in a terminal state unless the environment changes (new high-value tasks could steal resources, for example).

		A	B	C	D
Q	1	▲0!	▲		▲0!
(300)	2	▲			
R	1	▲		▲0!	
(100)	2		▲0!		

Figure 21: Final stage of marbles2 solution to the problem

Thus, in the end it has been determined that there is no competition for resources at all – all tasks can be satisfied with the available resources, using about 12 messages overall and about 4 message round-trips.

Experience and Limitations of Marbles2

As is evident from the curves in the Evaluation section below, this Marbles scheme (the first one written) tends to produce the lowest-quality solutions and also require largest number of messages. We believe that the latter is true because tasks bid on all qualified resources for every requirement, and in addition the scheme bids down prices one by one in epsilon increments (rather than in logarithmic sizes as some of the schemes below do). We have not had the time to investigate why the former is true.

Msmarbles [*allocation*: distributed, *elimination*: distributed]

In the Msmarbles (Multi-Sized Marbles) scheme each task has the same number of marbles. The size of each marble is the total value of the task divided by the number of marbles that the task has. Consequently, tasks with higher value have larger marbles.

Message Protocol

Tasks bid on resources by placing marbles on them. A task can bid one marble at a time, and must wait for a price-update message from the resource before placing another marble. Resources grant themselves to the task that has placed the largest value (not largest number) of marbles on them. When a task runs out of marbles, it can withdraw its marbles from a resource. When it does so, the resource returns all marbles to all tasks that have bid on it, with one exception. The resource keeps one marble from the current winner. In essence, the price for the current winner goes down to one marble.
When a task withdraws its marbles from a resource, it will not attempt to bid on that

resource again unless it has available at least one more marble than it got back. We call this number of marbles the task's "block amount" on a given resource. Block amounts always go up, and eventually will reach the point where a task cannot win an allocation of resources for all its requirements because the block amounts on the required resources exceeds the total number of marbles that a task has. When this happens, the task voluntarily withdraws from competition by withdrawing all marbles from all resources. The scheme converges because tasks keep withdrawing until all remaining tasks succeed. The intuition behind Msmarbles is that if the valuation of resources emerges incrementally, in small steps, it will be more accurate. This will enable tasks to make more informed decisions about where to place or withdraw marbles and when to give up, and thus lead to a better solution.

The timing of withdrawals is critical. It is advantageous to delay withdrawals as long as possible because by that time other tasks may have withdrawn first and hence they become subject to the eventually deadly block-amounts. In order to diminish the advantages of delays, we made each task have the same number of marbles, each task bid a single marble at a time, and each task wait for a reply before bidding the next marble. Richer tasks will have an advantage, as they should, because they can delay placing marbles. Poorer tasks may need several bids to catch up to the bid of a richer task, hence allowing the richer task to hold on to its marbles for a longer time.

One of the main qualities of Msmarbles is that multiple medium-sized tasks can together bid up the valuation of multiple resources forcing a richer task to become subject to several block amounts, and eventually forcing it to give up. This enables the Msmarbles scheme to make trade-offs between multiple medium-sided tasks and few richer tasks.

The Running Example under Msmarbles

Figure 22 shows the behavior of Msmarbles in the simple running example. In this example we gave each task 8 marbles (twice the number of resources). Task Q's marbles are worth 37.5 points, whereas Task R's marbles are worth 12.5 points. Lines labeled A, B, C and D represent the valuation of resources A, B, C and D over time. Lines Q-A, Q-B and Q-C represent the amount task Q has bid for resources A, B and C. R-A, R-C and R-B represent task R's bids. Initially, both tasks bid on A. Then they bid on the next resource they need: Q bids on B and R bids on C. When responses come back, Q learns that it is winning both resources. Task R learns that it is losing on A and winning on C. Task R must now bid for B, its only choice for requirement 2, and it keeps placing marbles on it until it outbids task Q. When Q is outbid it determines that the price increment to win D is 0+ (i.e., any amount larger than 0), and hence places a marble on D. At this point, both tasks are fulfilled and they stop bidding.

The second graph shows a more complex example where not all tasks can be fulfilled, and tasks need to withdraw bids and eventually withdraw from competition. The graph shows the evolution of the price for resource A, the amount task R bids on A (R-A), and the amounts task S bids on resources A, B and C (S-A, S-B and S-C respectively). In this example there are 4 tasks and 8 resources (not all shown in the graph), and S is the poorest task with 60 points. The graph shows how S first went on a bidding war for resource C and eventually withdrew because it needed marbles to bid on other resources. Similarly, S had losing bidding wars for resources A and B. A, B and C were the only choices that S had to fulfill one of its requirements, and after the three withdrawals, the

block amounts went so high that S would have had to use all its marbles to win one of those resources, leaving no marbles to win resources for its other requirements. At that point, S gave up, enabling the other three tasks to succeed. The price for A went down sharply enabling the task that needed it to use its marbles for other resources. The second problem comes from an example that Walsh uses to demonstrate that simple auctions cannot be used to compute optimal resource allocations when complementarities are present. For this particular example, but by no means for all, Msmarbles computes the optimal solution.

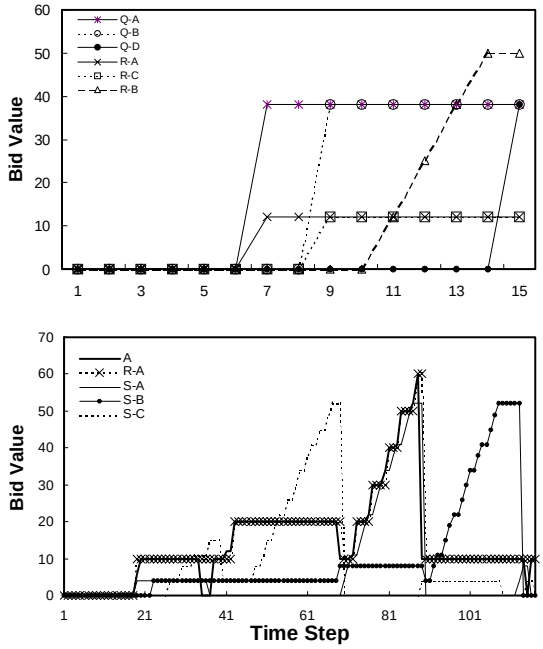


Figure 22: Bid values sequences for the Msmarbles scheme

Experience and Limitations of Msmarbles

The Msmarbles algorithm has not been as thoroughly evaluated as the others, so that implementation bugs disqualify it from the systematic comparison with the other algorithms in the Evaluation section. The solutions of the examples it does run are of high quality (defined as "close to the best solutions of other schemes"). However, the scheme is also one of the slowest, using significantly more messages than the others.

Marblesize [*allocation*: distributed, *elimination*: distributed]

The motivation of the Marblesize scheme is to allow trading off the quality of the solution against the number of messages needed through different pre-specified Marbles "sizes".

In the Marblesize scheme, no resource is free and the price for a resource is determined by the current highest bid. To acquire a resource, a task needs a certain number of marbles. Marbles have given size that can be subdivided in equal parts an arbitrary

number of times. The size of the marbles represents the minimum amount a task can bid on a resource. For each task, the initial marble size is equal to the task value divided by the number of requirements in that task. At the beginning, each task selects a possible combination of resources for its requirements and bid one marble on each of them. After that the bidding mechanisms continues based on the following rules: (1) If a task has more than one possible combination of resources, it chooses the cheapest one based on the current bids on those resources and allocate all its value among them but placing at least one marble on each resource. (2) A task wins if it is winning on all of its current bids. (3) A task loses if it is losing on all of its current bids. (4) A task that is winning on some of its bids can move one marble at a time from a winning resource bid to a losing resource bid. (5) A task can cut its marble size until the marble size is less than the minimum marble size allowed. (6) A losing task tries another resource combination and repeats the process. If it cannot find a new combination of resources it commits suicide.

Message Protocol
Task to resource: bid (amount), withdrawal (); Resource to task: loss (), win ().

The Running Example under Marblesize

First round: Q: Marble size (150) R: Marble size (50).

		A	B	C	D
Q (300)	1	▲	▲150!		▲
	2	▲150!			
R (100)	1	▲50		▲	
	2		▲50		

Figure 23: First stage of marblesize solution to the problem

The two requirements of task Q are winning so no changes happen in that task. In task R both requirements are losing. Since the first bidding proposal is no good it tries a second bidding proposal [C,B] while keeping a marble size of 50.

Second round: Q: Marble size (150) R: Marble size (50).

		A	B	C	D
Q (300)	1	▲	▲150!		▲
	2	▲150!			
R (100)	1	▲		▲50!	
	2		▲50		

Figure 24: Second stage of marblesize solution to the problem

Now R is winning on C that nobody wants and tries to move its marbles from C to B.

Third round: Q: Marble size (150) R: Marble size (25).

R cuts it marble size to the minimum size of 25. Although R is still winning on C, it cannot move its marble anymore because each resource needs at least one minimum size marble. R's second proposal is declared dead. Since it cannot try a third proposal, R is declared dead and the process terminates. Thus, the scheme fails to find the optimal solution for this simple problem - nevertheless it is the single best scheme we have for large problems, as will become evident in the Evaluation section.

		A	B	C	D
Q (300)	1	▲	▲150!		▲
	2	▲150!			
R (100)	1	▲		▲25!	
	2		▲75		

Figure 25: Final stage of marblesize solution to the problem

Experience and Limitations of Marblesize

The Marblesize scheme has the unique ability to trade off solution quality against speed of convergence. Figure 26 shows the impact that the minimum marbles size has on the total number of messages and the quality of the solution. As is evident, it is possible to control the minimum marble size to trade-off solution quality for computational time. In this example, an increase of less that 1% of solution quality is paid by a 10 fold increase in the number of messages.

Number of Subdivisions	Total Number of Messages	Maximum Value of Solution
4	7986	19017
3	6013	18950
2	3635	18894
1	1250	18880
0	793	18649

Figure 26: Trade-off between message traffic and solution quality in msmarbles

In terms of scalability with respect to problem size, the number of messages and solving shows a phase transition behavior where, for fixed number of resources, the number of messages increases sharply with the number of tasks until it reaches a certain value where it starts decreasing again, resembling the critical behavior observed in other combinatorial problems. We believe that this is due to the fact that for large number of tasks the lack of resources leads to quick suicide of most tasks with large requirements, thus the competition quickly decreases along the process.

Grabmarbles [*allocation*: distributed, *elimination*: distributed]

Grabmarbles is a variation of the Marblesize scheme which relies on heuristic selection of resource combinations. As in Marblesize, a task bids on the cheapest set of resources that will satisfy its requirements. Unlike the Marblesize scheme, re-bidding is not permitted after a losing resource bid, and bids are not based on marble sizes. Instead, a task agent submits a bid that is a heuristic evaluation of the task, based on its domain

value, number of task requirements, and number of alternative resources. A task only bids for resources whose prices (the evaluations of the currently winning tasks) are less than the bidding task's own evaluation. When a task agent loses a bid, it gives up on the current resource set and tries another if possible. The heuristics used by Grabmarbles were originally applied to Marbles2, and improvements in solution quality motivated the application of those heuristics to Marblesize.
A heuristic task evaluation function is defined for a given task and resource. (Note that this heuristic function actually violates the "no inflation" rule for Marbles schemes, making it impossible to use the prices paid for resources as an indication for their contribution of domain value. This has not been an issue because we have only measured pure solution quality so far.) The following example of a task evaluation function rewards tasks that have only one or two alternative resources to choose from, otherwise penalizing the task according to its number of requirements.

```
function taskeval (dval, reqs, alts)
  if alts = 1 return dval / reqs;
  else if alts = 2 return dval / (2 * reqs);
  else return dval / (4 * reqs);
```

Message Protocol
Task to resource: bid (amount), withdrawal (); Resource to task: loss (), win ().

The Running Example under Grabmarbles
First round: Q selects A and B.

		A	B	C	D
Q (300)	1	▲	▲37.5!		▲
	2	▲150!			

Figure 27: First stage of grabmarbles solution to the problem

The running example is analyzed here using the task evaluation function described above. All resources are initially free, so task agent Q selects A and B. Q's domain value of 300 and its 2 requirements yield an evaluation of 37.5 for resource A, while its evaluation with respect to A (150) reflects the fact that A is Q's only alternative resource for requirement 2.

Second round: R selects B and C.

		A	B	C	D
R (100)	1	▲		▲25!	
	2		▲50!		

Figure 28: Second stage of grabmarbles solution to the problem

The possible resource sets available to task agent R are (A,B) and (C,B). The cheaper alternative is (C,B), whose total price of 37.5 is due to Q's currently winning bid. Like task Q, R's second requirement has only one qualified alternative, so R's task evaluation with respect to resource B comes to 50. Task Q is outbid for resource B, so it withdraws

its bids and tries another resource combination.

Third round: Q selects A and D.

		A	B	C	D
Q (300)	1	▲	▲		▲37.5!
	2	▲150!			
R (100)	1	▲		▲25!	
	2		▲50!		

Figure 29: Final stage of grabmarbles solution to the problem

Task agent Q finally selects price-free resources A and D. Both tasks are now satisfied, reaching the optimal solution domain value of 400, with 15 messages passed.

Experience and Limitations of Grabmarbles

The Grabmarbles scheme produces solutions that are comparable to those of Marblesize, with a relatively small number of messages. The use of heuristics in evaluating each task's "deservedness" with respect to different resources has a globally beneficial effect on resource allocation. In the Marblesize scheme, the relative merit of competing tasks is resolved through the process of re-bidding and transferal of funds between resources. In Grabmarbles, the selection of resources through heuristics tends to direct the task agents toward resources they can realistically attain, while avoiding resources that are critical to other tasks. The focus on globally beneficial resource selection helps to eliminate the need for re-bidding.

The choice of task evaluation formula used in Grabmarbles has not yet been automated. The quality of solutions is greatly affected by how well suited the evaluation formula is for a particular problem set. The results shown in the curves in the Evaluation section were obtained using the following evaluation function.

```
function taskeval (dval, reqs, alts)
  return dval / reqs – 2 * alts;
```

This evaluation formula fails to yield the optimal solution domain value for the running example problem. The previous formula emphasizes the lack of resources available to a task, while the above formula only uses this as a tie-breaker. A hybrid evaluation formula, combining features of the two shown, has produced good solutions to all of these problem sets. But there remains a need for the automatic selection of an appropriate formula for a given problem, based on the distribution of task requirements per task, and alternative resources per requirement.

Brute-Force [*allocation*: centralized, *elimination*: centralized]

A trivial centralized brute-force solver that enumerates all possible solutions and then picks the best one was built. It is impractical for more than about 15 tasks and 30 resources but serves its purpose in producing small-size challenge problems for the Marbles schemes for which the optimal solution is known.

Random [*allocation*: centralized, *elimination*: centralized]

Similarly, we have built a solver which synthesizes a random solution, keeps it if it beats the previous one, and keeps doing this until it exceeds a given time limit. We have used it to establish lower bounds on the solution quality for large-size problems.

Simulated Annealing [*allocation*: centralized, *elimination*: centralized]

We have implemented a Simulated Annealing (SA) solver [Kirkpatrick 1983] to further compare the results of the different Marble solvers against well-known central schemes. The SA algorithm seeks to escape local maximum by accepting downhill moves with a probabilistic model based on statistical mechanics. In our implementation of SA we start by randomly assigning resources to tasks until all resources are allocated. Then, for a number of maxFlips times, we perturb or flip the state of the system to a neighboring state by randomly picking a task, a requirement from that task and a new resource for that requirement from its list of eligible resources. We evaluate δ, the change in the total value, and always accept the move if $\delta \geq 0$. If $\delta < 0$, we accept the move with probability $\exp(\delta/T)$, where T is the temperature parameter. We repeat this procedure for different values of T, starting with a high value of T and decreasing it following a geometric scheduling such that Ti+1= 0.5*Ti.

Experience and Limitations of the SA implementation

In terms of performance the SA solver ranks very close to but actually below the Marblesize solver. In certain problem instances SA beats Marblesize in finding a higher value in comparable execution size but on average Marblesize beats SA. SA provides the maximum number of flips (maxFlips) as its mechanism for externally controlling or trading-off quality of solution for execution time, similar to Marblesize using marble granularity for the same purpose. Even for a surprisingly low values of maxFlips, SA finds solutions within a few percent of the highest value with a significant speed up in solution time. With such a low value of maxFlips, SA is our "most efficient" solver (as measured by dividing solution quality by running time).

SAT Encoding [*allocation*: centralized, *elimination*: centralized]

A centralized SAT solver was implemented by encoding the resource allocation problem into Boolean satisfiability formulas in conjunctive normal form (CNF). In this approach, the allocation of resources to tasks is obtained by finding truth assignments to the resulting formulas. To use satisfiability testing for optimal allocation of resources we turn to the problem of finding valid assignments of resources for at least k (with $k \leq N$, the total number of tasks) tasks and then do a binary search to find the maximum k. This problem can then be encoded into a CNF formula of the following form:

$$f = f_k \wedge f_{cross} \wedge f_1 \wedge f_2 \wedge \ldots \wedge f_N$$

Where fk is responsible for switching on at least k of the variables representing the N tasks, fcross precludes resources from being assigned to more than one requirement and if (i=1,2,…,N) selects eligible resources within each individual task.

SAT encoding of the running example
To encode the running example presented above for at least two tasks (k = 2) filled we define the following 13 boolean variables. First we introduce the tasks variables: t1 and t2, that represent each task in the formula. Then we define the resources variables A11, A12, A21, B11, B21, C21 and D11. Where Aij=TRUE indicates the assignment of resource A to task i requirement j. To select at least 2 different tasks variables we introduce four additional variables p1, p2, r1, r2 with the condition that p1 → -r1, p2 → -r2, (p1,r1) → t1 and (p2,r2) → t2. With this variables definition, the formulas introduced above take the following form:

$$f_{k=2} = (p_1 \vee p_2) \wedge (r_1 \vee r_2) \wedge (\bar{p}_1 \vee \bar{r}_1) \wedge (\bar{p}_2 \vee \bar{r}_2) \wedge (\bar{t}_1 \vee p_1 \vee r_1) \wedge$$
$$(t_1 \vee \bar{p}_1) \wedge (t_1 \vee \bar{r}_1) \wedge (\bar{t}_2 \vee p_2 \vee r_2) \wedge (t_2 \vee \bar{p}_2) \wedge (t_2 \vee \bar{r}_2)$$

$$f_{cross} = (\bar{A}_{11} \vee \bar{A}_{12}) \wedge (\bar{A}_{11} \vee \bar{A}_{21}) \wedge (\bar{A}_{12} \vee \bar{A}_{21}) \wedge (\bar{B}_{11} \vee \bar{B}_{22})$$
$$f_1 = (\bar{t}_1 \vee A_{11} \vee B_{11} \vee D_{11}) \wedge (\bar{t}_1 \vee \bar{A}_{11} \vee \bar{B}_{11}) \wedge (\bar{t}_1 \vee \bar{A}_{11} \vee \bar{D}_{11}) \wedge$$
$$(\bar{t}_1 \vee \bar{B}_{11} \vee \bar{D}_{11}) \wedge (\bar{t}_1 \vee A_{12})$$
$$f_2 = (\bar{t}_2 \vee A_{21} \vee C_{21}) \wedge (\bar{t}_2 \vee \bar{A}_{21} \vee \bar{C}_{21}) \wedge (\bar{t}_2 \vee B_{22})$$

We solve the resulting formula f using a Java implementation [Jackson] of the WSAT [Selman 1993] solver. One can verify that f evaluates to TRUE by setting A12, B22, C21, D11, t1, t2, p1 and r2 to TRUE and all other variables to FALSE, which yields the correct solution for the running problem.

Experience and Limitations of the SAT Encoding
Our current SAT-based solver performs very well compared to Marblesize and Simulated Annealing for small and medium size problems (i.e., N ≈ 50). For larger problems (e.g., N=100) the solution time degrades about an order of magnitude compared to Marblesize and SA but it is still able to produce high value results. By controlling the number of solutions that we ask WSAT to generate, we can externally trade-off solution quality with execution time and the solver can sometimes find solutions within less than 5% of the best value found with SA but with 10 to 20 times speedup. Another advantage of this approach is that it can be used to rapidly estimate the maximum number of filled tasks without having to search for the optimal solution. In its current implementation the SAT-based solver is fully centralized but the same SAT encoding approach can be combined with Marbles or other distributed market mechanisms [Walsh 1998] to produce a distributed solver.

3.7.6 Evaluation

We evaluated the performance of the different solvers described above on synthetic problems that have the same characteristics of the problems stated above but with arbitrary number of resources and tasks. The problems were generated by randomly assigning to each task a certain number of requirements and a task value. The set of

possible resources for each task was also randomly selected from the original resource pool. These random values were independently selected from three different Gaussian distributions. Thus, the dominant parameters in describing a given problem are: a) number of tasks, b) number of resources, c) r, average number of requirements per tasks, d) v, average task value and e) p, the average number of possible resources per requirement.

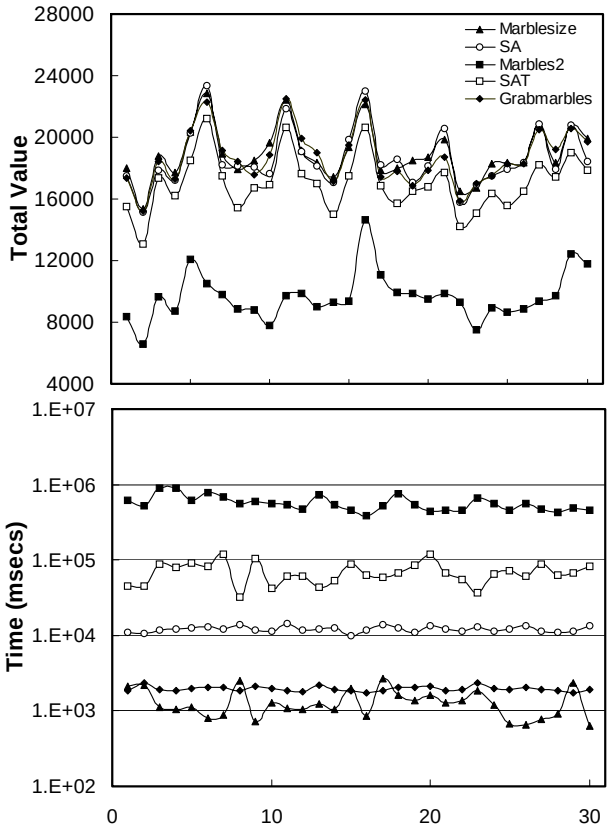


Figure 30: Quantitative Comparison of MARBLES Algorithms

Figure 30 compares the performance of our solvers for 30 different problems with 100 resources and 100 tasks. The problems were generated with r =4, v = 300 and p = 10. The parameters we use to evaluate performance are the total value (i.e., the sum of the task values for all filled tasks) of a solution and the (execution) time it took the solver to find that solution. In Figure 1a and b, we compare the results for total value and time, respectively. Marblesize, Grabmarbles and Simulated Annealing can find comparable results of the total value but with Marblesize being 3 to 4 times faster than Grabmarbles and about an order of magnitude faster than Simulated Annealing. The results obtained with SAT and Marbles2 are of lesser quality in terms of performance but we see that they follow the same structure found in the other curves suggesting that all curves are somehow converging towards an optimal solution.

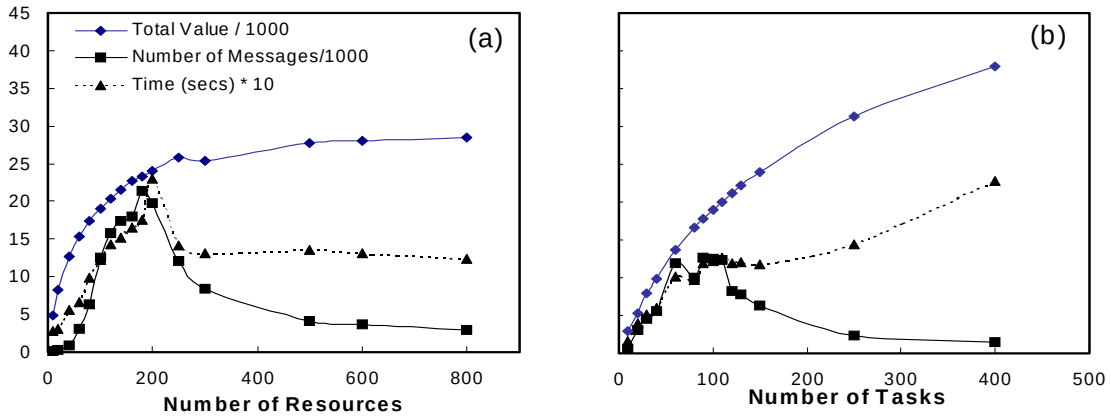


Figure 31: Easy-hard-easy phase-transition behavior of the total number of messages and computational time for the Marblesize scheme. (a) 100 tasks, (b) 100 resources

Figure 31 shows the behavior of the total number of messages, execution time and total value of solutions found with Marblesize for different size of the problem. In Figure 31a, shows results for 100 tasks and different number of resources while in Figure 31b the results correspond to 100 resources and different number of tasks. In both curves we observe an easy-hard-easy phase-transition effect where the number of messages (and time) increases very drastically as the problem gets larger until it reaches a peak and after that drops down again. This property of Marblesize is due to the fact that unlikely to succeed tasks drop out of the competition very early in the process and do not waste any bidding messages. Since the distributions of task values and number of requirements per tasks are independent, tasks with large number of resources and low task value end up with marbles of relatively small size that makes them lose in all bids before entering competition. In Figure 31b, the execution time continues to rise slowly after the transition peak while the number of messages drops down and this is due to the fact that in the initial phase tasks need to evaluate alternative combination of resources before bidding and the total computational time of this operation increases with the number of tasks.

3.7.7 Related Work

It is desirable to obtain distributed negotiation schemes in which (1) domain experts can understand the decisions made by negotiation participants because they use a domain currency for making their trade-offs that the experts share, and that (2) can be "steered" in its collective real-time response, fault tolerance, and solution quality behavior by changing their relative desirability at run-time.

We list the most relevant non-market-inspired previous work on distributed resource allocation in the References section, but do not have the space here to discuss them at any length (they generally address neither (1) nor (2) above). Instead, we will use the

remaining space to put the auction protocols we use for resource acquisition in the context of prior work. A negotiation protocol in our terms defines the types of messages that can be sent and how they can be strung together (the syntax of message exchange). In our terminology, this – together with the bidding strategies of requesters and the auctioning strategy of requesters – defines a "scheme" for market-based distributed resource allocation.

Walsh et al. (1998) outline the fundamental choices in this design space: (a) single-resource auctions, (b) combinatorial auctions, and (c) Vickery auctions. Combinatorial auctions as generally inapplicable to truly distributed assignment of resources. This is because they need a large number of messages to coordinate between themselves (as they cannot individually auction themselves but must bundle up with others to be bid on in combination). In (c) Vickery auctions, every resource requester has an incentive to report his true requirements to a centralized auction mechanism which can then make an optimal assignment of resources (solving an NP-complete problem) and report the assignments back to the requesters. This is obviously not an option for truly distributed resource allocation either, and was not investigated further.

This leaves (a) single-resource auctions, in which each resource can auction itself off to the highest-value task based solely on its local bid information. Our Marbles schemes are a subclass of single-resource auction. However, the distributed algorithms introduced for the altruistic task suicide phase further distinguish our Marbles schemes from work on competitive auctions. This task suicide phase is fundamental for the quick convergence of the Marbles schemes: by lowering resource prices it usually helps other tasks succeed.

3.7.8 MARBLES Conclusion

It is too early to make any claims on how far from "optimal" in any sense the currently implemented Marbles schemes are (be that in term of the quality of the solution, in terms of the number of messages needed, or any combination thereof). However the following conclusions can be drawn:

1. Marbles-type distributed collaborative negotiation schemes are an exciting and worthwhile research program for years to come; this is because there are many "optimal" solvers depending on how much the application domain values fault-tolerance, average response time, real-time response guarantee, and quality of the solution.

2. "Phase transitions" were seen as is evidenced in Figure 31; to be precise, there were Gauss-like curves for the amount of messages needed based on a varying number of resources for a fixed number of tasks. A Marbles scheme finds out quickly that few tasks can be satisfied with the very few resources, as well as that nearly all tasks can be satisfied with the abundant resources, but uses substantially more computation if there are "just enough resources for most of the tasks with the right assignments". However, currently there was no way of predicting how much negotiation a given problem requires.

3. It seems that the Marbles schemes with good performance all seem to have the

property of eliminating (apparently) losing tasks very early on.

4. As this is work in progress the schemes were not compared against other distributed algorithms at great length. However, based on our performance comparisons of our best Marbles schemes to the well-known centralized Simulating Annealing strategy it is likely that this family of market-inspired collaborative negotiation schemes is well-suited to the real-time distributed solution of resource allocation problems.

4 Transitions to Military Applications

The main deliverable of the CAMERA project was the SNAP flight scheduling application that embodies the CAMERA technology that was fielded and used by the US Marines. The system was fielded at the following locations:

Marine Air Group 13 in Yuma. The system was fielded in all 4 squadrons.

Marine Expeditionary Units. The system was fielded on board the USS Bonhomme Richard, the USS Belleau Wood, the USS Pelleliu and the USS Essex that conducted operations in Iraq, Japan and Afghanistan.

The software produced by the CAMERA project involves over 500,000 lines of code. Of these about 40% is part of the generic negotiation framework, and 60% is application specific.

Currently there is ongoing collaboration with Lockheed-Martin Information Systems (Orlando, Florida) to also transition the CAMERA-funded flight scheduling software to the Joint Strike Fighter program (which will eventually replace the current Harrier fleet), and are integrating our software with Lockheed-Martin's autonomic logistics effort. Lockheed-Martin demonstrated a first prototype incorporating our software in early 2004.

5 The ANTS E-Book

The project also produced the DARPA Autonomous Negotiating Teams Electronic Book, an archival compendium of research funded by this and other projects funded by the DARPA ANTs Program:

http://www.isi.edu/~szekely/antsebook/ebook/

The e-book was edited by Alejandro Bugacov, Stephen Fitzpatrick, Gabor Karsai, Victor Lesser, H. Van Dyke Parunak, Vijay Raghavan, Bart Selman, and Pedro Szekely, and contains detailed problem descriptions for the two challenge problems addressed by the ANTs program, one in logistics and one in electronic warfare. It also contains technical descriptions of the scheduling, distributed constraint optimization, Bayesian tracking, case-based reasoning, and mediation techniques for negotiating teams that were investigated by ANTs contractors, and points to re-usable software that resulted from the effort.

6 References

Andersson, M., and Sandholm, T. 1999. Time-Quality tradeoffs in reallocative negotiation with combinatorial contract types. In Proceedings of AAAI-99, Orlando, Florida.

Atkins, E.; T. Abdelzaher; Shin, K.; and E. Durfee, E. 1999. Planning and resource allocation for hard real-time. In Proceedings of the Third International Conference on Autonomous Agents, Seattle, Washington.

Boutilier, C.; Goldzmidt, M.; and Sabata, B. 2000. Sequential auctions for the allocation of resources with complementarities. In Proceedings of IJCAI-99, Stockholm, Sweden.

Chen, J.; Bugacov, A.; Szekely, P.; Frank, M.; Cai, M.; Kim, D.; and Robert Neches, R. Distributed Resource Allocation: Knowing When To Quit. Accepted at the Representations and Approaches for Time-Critical Decentralized Resource/Role/Task Allocation Workshop of the Second International Joint Conference on Autonomous Agents & Multi-Agent Systems (AAMAS 2003). Melbourne, Australia.

Choy, M., and Singh, A. 1992. Efficient fault tolerant algorithms in distributed systems. In 24th ACM Symposium on Theory of Computing, pp. 593–602.

Collins, J.; Sundareswara, R.; Tsvetovat, M.; Gini, M.; and Mobasher, B. 1999. Search strategies for bid selection in multiagent contracting. In JCAI-99 Workshop on Agent-mediated Electronic Commerce (AmEC-99).

Ferguson, D.; Nikolaou, C.; Sairamesh, J.; and Yemini, Y. 1996. Economic models for allocating resources in computer systems. In S. Clearwater (Ed.), Market-Based Control: A Paradigm for Distributed Resource Allocation. Hong Kong: World Scientific.

Frank, M.; Bugacov, A.; Chen, J.; Dakin, G.; Szekely, P.; and Neches, R. "The Marbles Manifesto: A Definition and Comparison of Cooperative Negotiation Schemes for Distributed Resource Allocation," Proceedings of the 2001 AAAI Fall Symposium on Negotiation Methods for Autonomous Cooperative Systems, November 2-4, 2001, North Falmouth, Massachusetts.

Garey, M., and Johnson, D. 1979. Computers and Intractability: A Guide to the Theory of NP-Completeness. San Francisco, CA: W.H. Freeman and Company.
Gent, I., and Walsh, T. 1993. Towards an Understanding of Hill-Climbing Procedures for SAT. In Proceedings of AAAI-93, pp. 28-33.

Jackson. We used Dr. D. Jackson's Java implementation of WSAT available at http://sdg.lcs.mit.edu/walksat

Kirkpatrick, S.; Gelatt, C.; and Vecchi, M. 1983. Optimization by Simulated Annealing. Science, 220, 671-680.

Milgrom, P. 2000. Putting auction theory to work: The simultaneous ascending auction. Journal of Political Economy.

Sandholm, T., and Lesser, V. 1997. Issues in Automated Negotiation and Electronic Commerce: Extending the Contract Net Framework. Readings in Agents, pp. 66-73, Morgan Kaufmann.

Selman, B.; Kautz, H.; and Cohen, B. 1996. Local search strategies for satisfiability testing. AAAI-92, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, 26:521-532.

Smith, R. 1980. The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. IEEE Transactions on Computers 29(12):1104-1113.

Waldspurger, C., and Weihl, W. 1994. Lottery scheduling: Flexible proportional-share resource management. In Proceedings of the First Symposium on Operating Systems Design and Implementation, pp. 1–11.

Walsh, W.; Wellman, M.; Wurman, P.; and MacKie-Mason, J. 1998. Auction protocols for decentralized scheduling. In Eighteenth International Conference on Distributed Computing Systems, Amsterdam, The Netherlands.

Walsh, W.; Wellman, M. 1998. Market SAT: An extremely decentralized (but really slow) algorithm for prepositional satisfiability. In Seventh National Conference inartificial Intelligence, 303-309, 2000.

Wellman, M. 1996. The economic approach to artificial intelligence. ACM Computing Surveys 28 (4es):14–15.